Cloud Mobile System

CLOMOSY EĞİTİM İÇERİĞİ

Bulut Tabanlı Mobil Sistem

HIZLI UYGULAMA GELİŞTİRME

Aralık 2024

V1.2

https://clomosy.com/docs/ClomosyKitapV2.pdf

atiker yazılım

ÖNSÖZ

Kişisel masaüstü bilgisayarların gelişmesi, Windows gibi çok görevli ve kullanıcılı işletim sistemlerinin ortaya çıkmasına zemin hazırlamıştır. Bu sayede cep telefonları, yalnızca bir iletişim aracı olarak kalmamış, aynı zamanda yazılım geliştirme cihazları hâline gelmiştir. Bu durum, teknolojinin ilerlemesiyle birlikte IOS ve Android platformların gelişmesini daha da karmaşık hâle getirmiştir.

Clomosy (Cloud Mobile System) Mobil Uygulama Geliştirme Platformu, teknolojinin gelişmesiyle birlikte daha karmaşık hâle gelen bu ortamda, geliştiricilerin daha hızlı ve kolay bir uygulama geliştirme süreci yaşayabilmelerini sağlamak amacıyla ortaya çıkmıştır. Bu platformla birlikte, kodlama süresi azaltılabilir ve kod kalitesi artırılabilir hâle gelmiştir.

Geliştiricilerin birden fazla platform üzerinde ayrı ayrı kodlar yazarken yaşadıkları sorunlar ve katlandıkları maliyetler, işletmelerin hem IOS hem Android uygulamalar için farklı personel istihdam etme zorunlulukları göz önüne alınmış ve Clomosy'nin gelişerek bugünkü hâlini almasını sağlamıştır veya ortam oluşturmuştur.

Clomosy geliştirme ortamı bulut bilişim sistemleri üzerinde olduğu için uygulama geliştirmek hem son derece hızlıdır hem de herhangi bir dosya indirme işlemi veya kurulum zahmeti yoktur. Aynı zamanda bir işletim istemine de bağlı olmayıp geliştirici yer ve zaman fark etmeksizin herhangi bir işletim sistemi üzerinde uygulama geliştirebilir. Sürecin hızlı olması, geliştiricinin dikkatinin dağılmasını engeller ve sürükleyici bir ortam sağlar.

Bu kitapta yer alan örnek uygulamalar, yazılım geliştirme işinin teknik yönlerini ve unsurlarını öğretirken yazılımla ilgili problemlerin nasıl ortaya çıktığını da geliştiriciye anlatmış olur. Bu sayede geliştiriciye benzer problemleri çözebilme yeteneği kazandırır. Kitaptaki gerçek hayat problemlerine çözüm üreten uygulama örnekleri, yazılım sektörünün ne kadar geniş bir kullanım alanı olduğunu da geliştiriciye göstermiş olur.

Clomosy TRObject dili, yazılım geliştirmeye yeni başlayanlar için programlamayı deneyimleyerek öğrenmeye teşvik ederken aynı zamanda algoritma oluşturma becerisi de kazandırır. Bileşenlerinde yer alan teknolojiler sayesinde yazılım sektörünün farklı dallarında örnekler sunarak uzmanlaşma alanlarının daha iyi kavranmasına yardımcı olur.

Özetle, Clomosy TRObject programlama dili web üzerindeki geliştirme ortamında yazılan kodların aynı anda Android, IOS ve Windows üzerinde çalıştırılmasını sağlayarak geliştiricilere hızlı ve kolay bir yazılım geliştirme deneyimi sunar.

"Clomosy'e Hoş Geldiniz." diyerek kitabımızın ilk kısmına geçelim:

Gerek veri tabanı uygulamaları, gerekse diğer uygulamalarda (MQTT, IOT, cihaz donanımlarına erişim desteği vb.) gelişmiş projeler oluşturmak Clomosy ile oldukça kolay ve güvenlidir. Kitapta uygulama geliştirme alanında kullanabileceğiniz pek çok konuya değinilmiştir. Herhangi bir alanda uzmanlaşmak istediğinizde veya daha fazla teknik bilgiye ihtiyacınız olduğunda "docs.clomosy.com" adresini ziyaret edebilirsiniz.

İçindekiler

BÖLÜM 1. Clomosy Genel Yapısı

TRObject programlama dilini öğrenmek isteyen bir programcı için fonksiyonların nasıl yazıldığını ve nasıl kullanıldığını kapsamlı bir şekilde öğrenmek oldukça önemlidir.

Bir program yazılmaya başlandığında değişken tanımlamasından sonra ilk yapılması gereken programın ana akışını yönlendirmek için kullanılan main program bloğunu tanımlamaktır. Main bloğu tanımlanmadan kodları çalıştırdığınızda, programda bir şey görünmeyecektir. Bu blok yapısında diğer tanımlanacak her bir fonksiyon veya prosedür çağrılabilir ve yönlendirme işlemleri gerçekleştirilebilir.

Clomosy üzerinde programın akışını kontrol etmek için oluşturulan main program bloğu, şu şekilde oluşturulabilir:

```
{

…

}
```

Bu blok yapıları, koşullu ifadeleri ve döngüleri belirtmek için kullanılır. Aynı zamanda bir prosedür veya fonksiyonun başlangıcını ve sonunu işaretler.

1.1. Clomosy Dil Yapıları

Clomosy platformu, TRObject dil yapısını sunar. Bu dil, farklı programlama paradigmalarına uygun tasarlanmış, geliştiricilere çeşitli seçenekler sunmaktadır. İşte dil yapısının ana özellikleri:

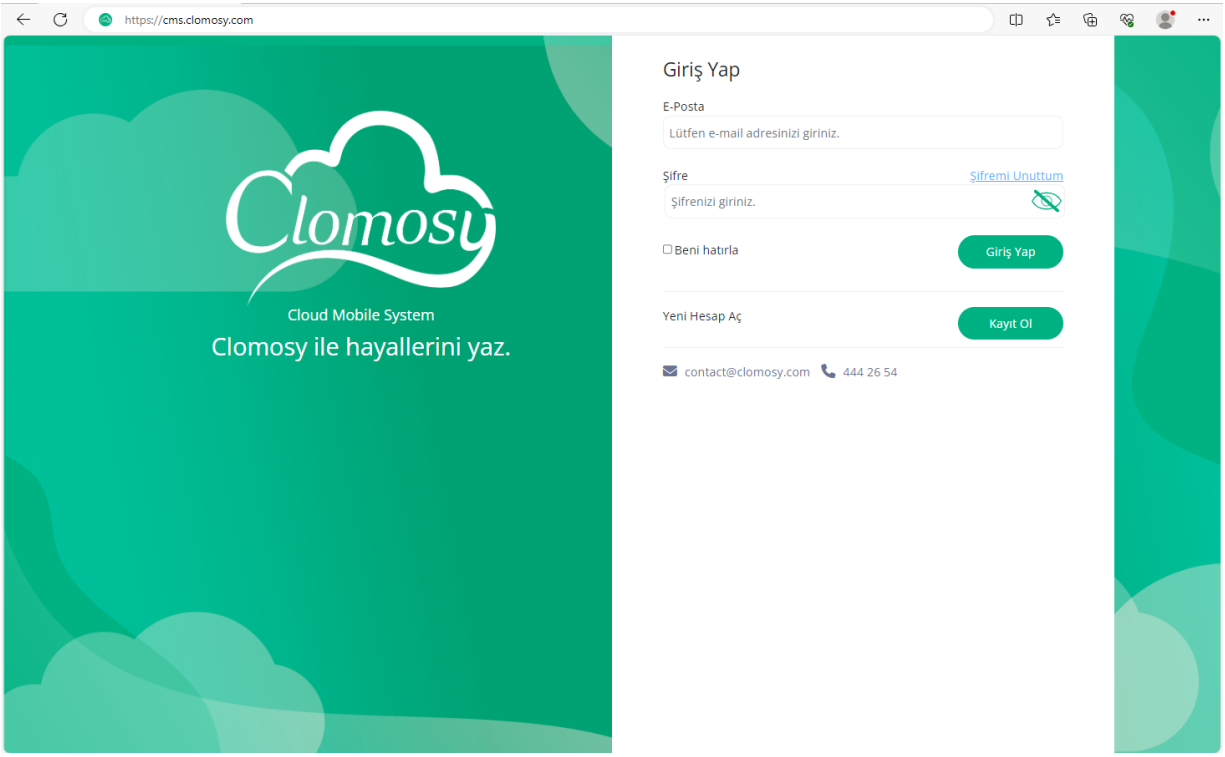
TRObject Dil Yapısı (.tro Dosya Uzantılı):

- TRObject, C benzeri süslü parantezli bir dil yapısına sahiptir.
- Dosyaları .tro dosya uzantısı ile tanımlanmaktadır.

1.2.Hesap Oluşturma

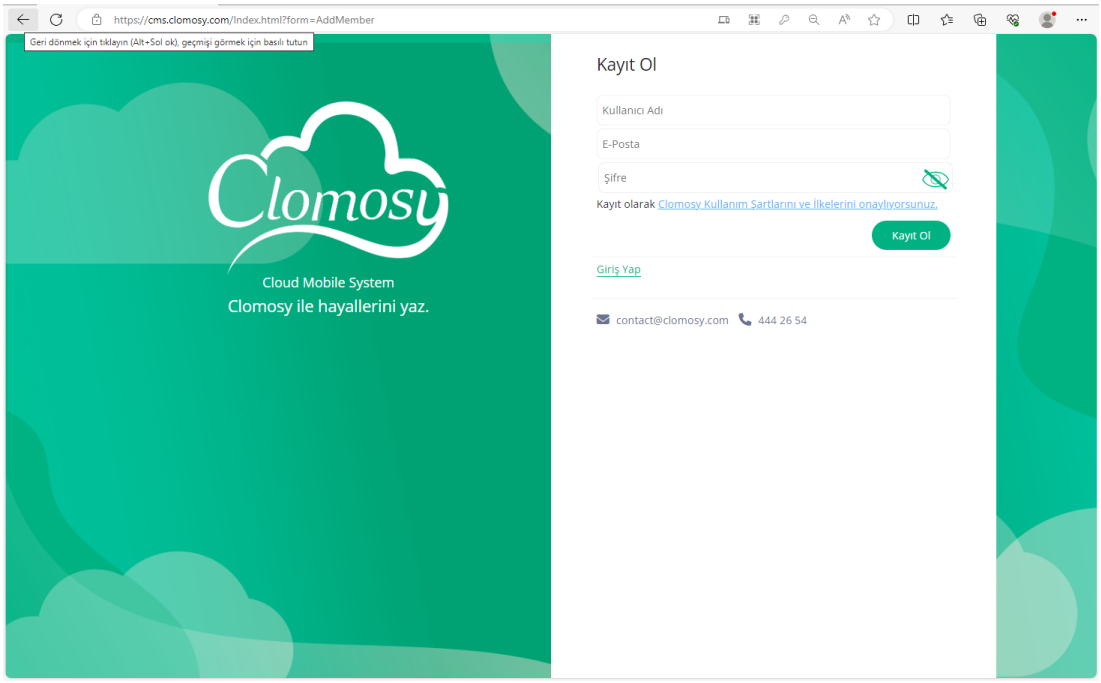
1.2.1. Geliştirici Hesabı Oluşturma

Kişi, Clomosy üzerinde geliştirici olmak istediğinde https://clomosy.com.tr veya https://clomosy.com sitesine giderek kayıt ol butonuna tıklayıp geliştirici sitesine yönlendirilecektir (https://cms.clomosy.com).



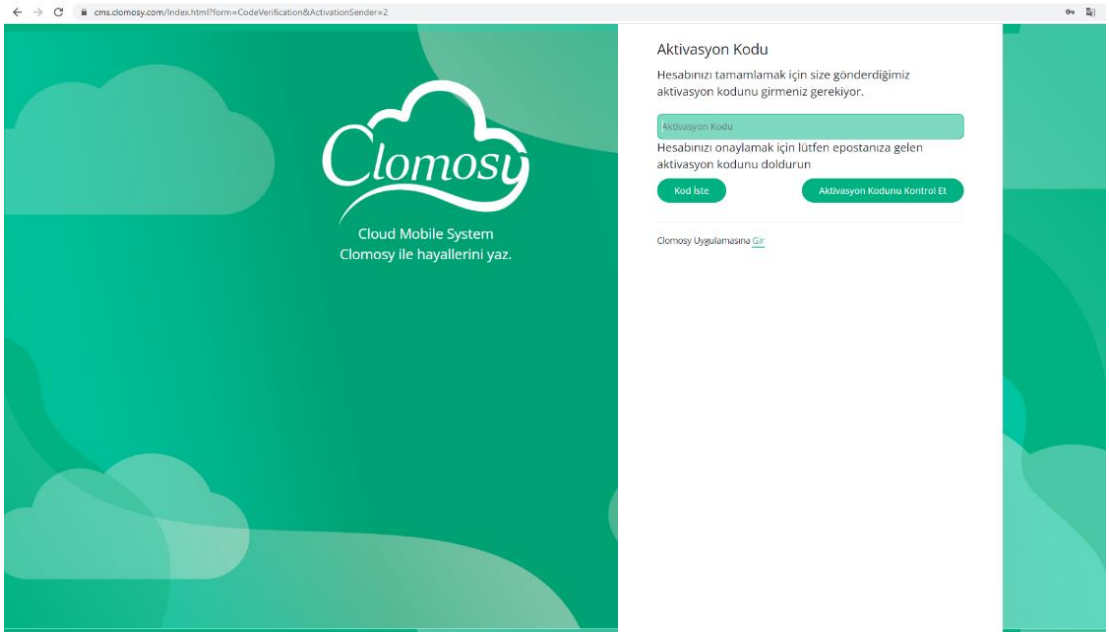
Resim 1

Kullanıcı, *"Kayıt Ol"* butonuna tıkladıktan sonra *Kayıt Sayfası* (Resim 2) gelmektedir. Burada kullanıcı adı, e-posta adresi ve şifre girmelidir. "*Kayıt Ol"* butonuna tıkladığında e-posta adresine aktivasyon kodu geleceği için e-posta adresinin doğru ve tam yazılmasına dikkat edilmelidir.



Resim 2

"Kayıt Ol" butonuna tıkladıktan sonra girilen e-posta adresine aktivasyon kodu gitmektedir. Web sitesinde ise *Resim 3*'te bulunan ekran gelir. Bu ekranda e-posta adresine gelen aktivasyon kodu yazılır.



Resim 3

Yukarıda aktivasyon kodu yazıldıktan sonra *Aktivasyon Kodunu Kontrol Et* butonuna tıklanıp *Giriş Yap* sayfasına yönlendirildiğinde, geliştirici hesabı kayıt işlemi başarıyla tamamlanmıştır. Ardından bilgileriniz ile giriş yapılabilmektedir.

Clomosy hizmetinin premium ve freemium geliştirici hesapları bulunmaktadır. Premium hesabı ücretli, daha geniş ve özel avantajlar sunan versiyonudur. Freemium hesapta ise bazı kısıtlamalar mevcuttur. Bu iki hesap arasındaki farklara bakalım.

Premium Hesabı

Premium hesabı özellikleri;

- Clomosy tarafından birden fazla firma verilmektedir.
- Sınırsız sayıda proje oluşturulabilir.
- Sınırsız Unit desteği sağlanır.
- Şablon (Template) desteği mevcuttur. Şablonlara *Yönetim Paneli* kısmından erişilebilir. Yönetim Paneli özellikleri konu başlığı altında anlatılmıştır.
- Proje okuma izni mevcuttur.

Freemium Hesabı

Premium hesabı gibi her özellik açık bir şekilde kullanılmamaktadır. Bu hesapta bazı özellikler kısıtlanmıştır.

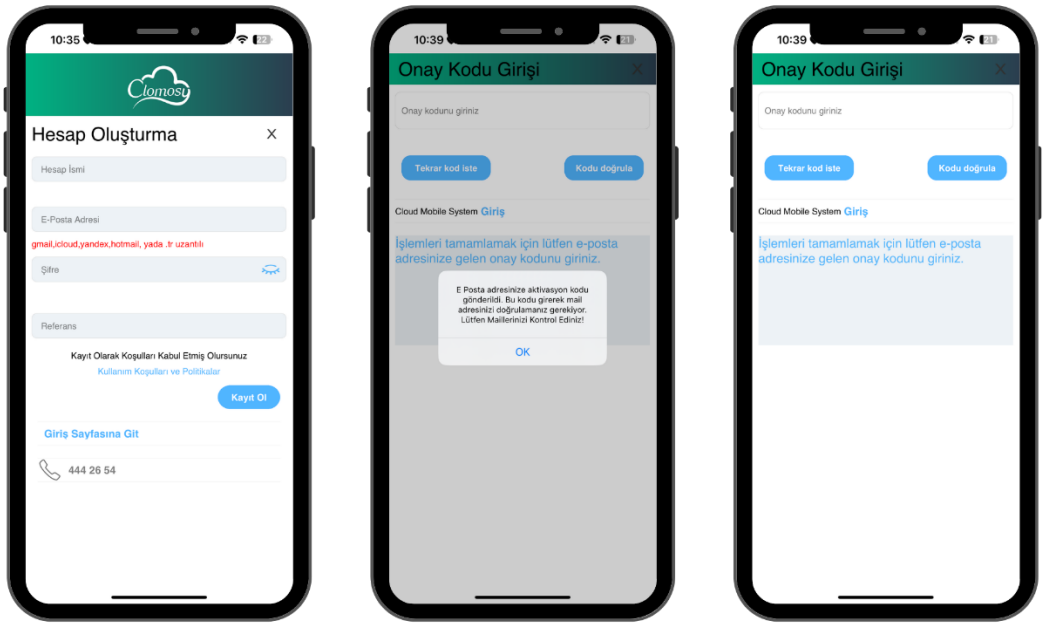
Freemium hesabı özellikleri;

- Tek bir firma kullanılmaktadır. Firma adı *Educational* olarak geçmektedir.
- Maksimum 2 proje oluşturulmaktadır.
- Maksimum 21 Unit açılmaktadır.
- Ana Kod ve Unit için uygulanan karakter sınırı 100 bin karakterdir.

1.2.2. Kullanıcı Hesabı Oluşturma

Clomosy platformunda geliştirilen projeyi görebilmek için kullanıcı hesabı oluşturulmalıdır. App Store, Google Play Store'dan "Clomosy Learn" uygulamasını indirerek mobil cihaz üzerinden uygulama açılmalıdır. "*Kayıt ol"* butonuna tıklayarak görseldeki *Hesap Olusturma* ekranına, hesap bilgileri yazılmalıdır. Bilgiler girilirken dikkat edilmesi gereken iki önemli konu bulunmaktadır. İlk konu; geliştirici hesabı ve kullanıcı hesabının e-posta adresinin farklı olmasıdır. İkinci konu ise; referans kısmının boş bırakılmasıdır. Referans kısmına Clomosy tarafından onaylanmamış proje aktivasyon kodu girilmemektedir. Bu nedenle referans kısmı boş bırakılmalıdır.

Bilgiler girildikten sonra "*Kayıt Ol"* butonuna tıklanır ardından *Onay Kodu Girişi* sayfasına geçilir. E-posta adresine gelen kod girilip "Kodu doğrula" butonuna tıklanır ve giriş sayfasına yönlendirilir. Artık yeni bir kullanıcı hesabı oluşturulmuştur.



Resim 4

Oluşturulan kullanıcı hesabı mobil platformlarda (Android ve IOS) kullanılacağı gibi Windows platformunda da ClomosyLearn.exe üzerinde kullanılabilir. ClomosyLearn.exe https://clomosy.com/win/ClomosyLearn.zip adresi üzerinden indirilebilir.

1.3. Proje Oluşturma

Geliştirici ortamına giriş yaptıktan sonra karşınıza, aşağıdaki görselde bulunan firma ekranı gelmektedir. Bu ekranda, sağ üst kısımda profil fotoğrafına tıklandığında "Profilim" ve "Çıkış Yap" sayfalarına yönlendirilir.

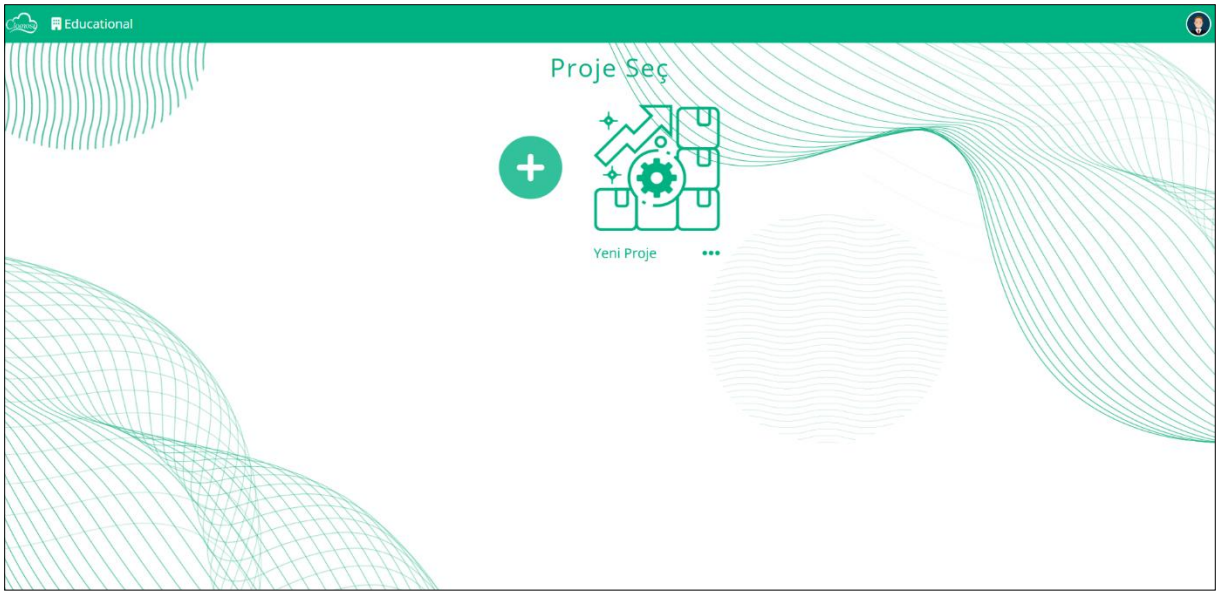
Sayfadaki "Eğitimler" butonuna tıklandığında alınan eğitimler yer alır. "Belgelerim" butonuna tıklandığında ise alınan belgelerin listesi yer alır. Burada görüntüle diyerek belgeye ulaşılır. İstenirse belge indirilebilir veya paylaşılabilir.

Proje sayfasına geçiş yapabilmek için firmaya tıklanması gerekir.

Resim 5

Bir firma içerisinde birden fazla proje olabilir. Bu projeleri oluşturabilmek için + (artı) butonuna tıklayarak yeni bir proje açmanız gerekmektedir. Proje adı girilir ve kaydedilir. Bir firma içerisinde birden fazla proje yer alabilir. Bu projeleri oluşturabilmek için + (artı) butonuna tıklayarak yeni bir proje açmanız gerekmektedir. Proje adını girdikten sonra kaydedebilir, proje ismini değiştirebilirsiniz. Değiştirme işlemini proje adının yanında bulunan üç noktaya tıklayarak gerçekleştirebilirsiniz. Projeyi silmek istediğinizde yine üç noktaya tıkladığınızda karşınıza çıkan çöp kutusuna tıklayarak silebilirsiniz.

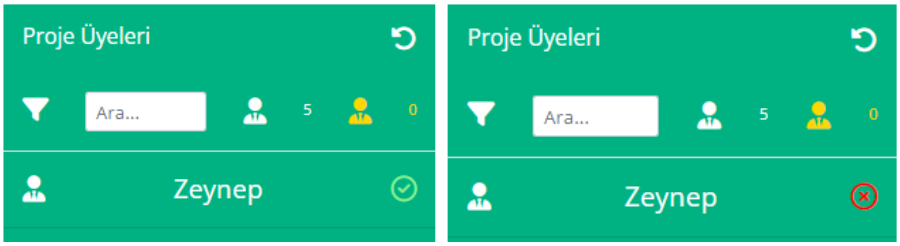
Sonrasında ise projeye tıklayarak geliştirici ortamına geçiş yapabilirsiniz.



Resim 6

1.4. Projeye Dahil Olma

Clomosy'de geliştirilmiş olan projeyi kullanabilmek için projeye dahil olmak gerekir. Kullanıcı, projeye dahil olma isteği gönderdikten sonra aşağıdaki geliştirici ekranına istek gelir, yeşil tik üstüne tıklayarak projeye dahil olma işlemi başarıyla gerçekleştirilir. Bu durumda artık üye, projenize dahil olmuş demektir.



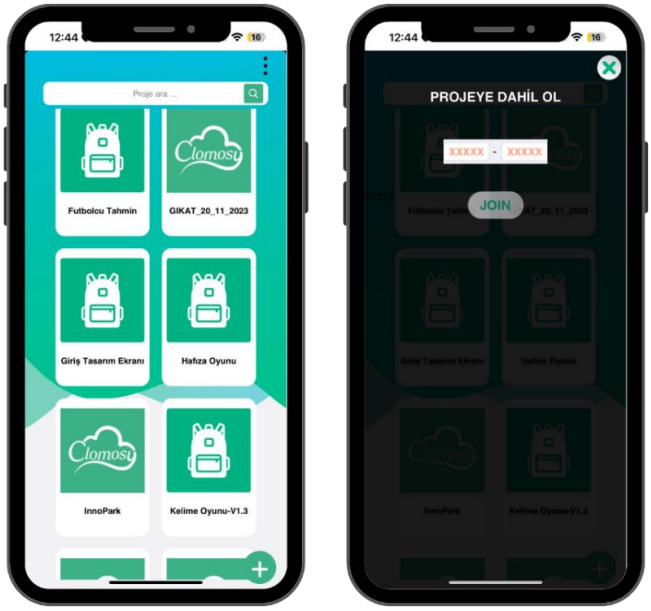
Resim 7

Dahil olma işleminin birden fazla yolu bulunmaktadır. Aşağıda bu yöntemler anlatılmaktadır.

1.4.1. Projeye Kullanıcı Ekranından Dâhil Olmak

Bu yöntem, kullanıcıların mobil uygulama üzerinden, geliştirici hesabında açılan projeye dahil olmasını sağlamaktadır. Burada dikkat edilmesi gerekilen önemli nokta; projeye dahil olmak için hesapta proje sayısı birden fazla olmalıdır. Kullanıcı, uygulama açıldığında proje listesinde, sağ alt köşede bulunan + (artı) butonuna tıklamalıdır. Karşısına çıkan ekranda proje aktivasyon kodunu yazarak projeye dahil olabilir.

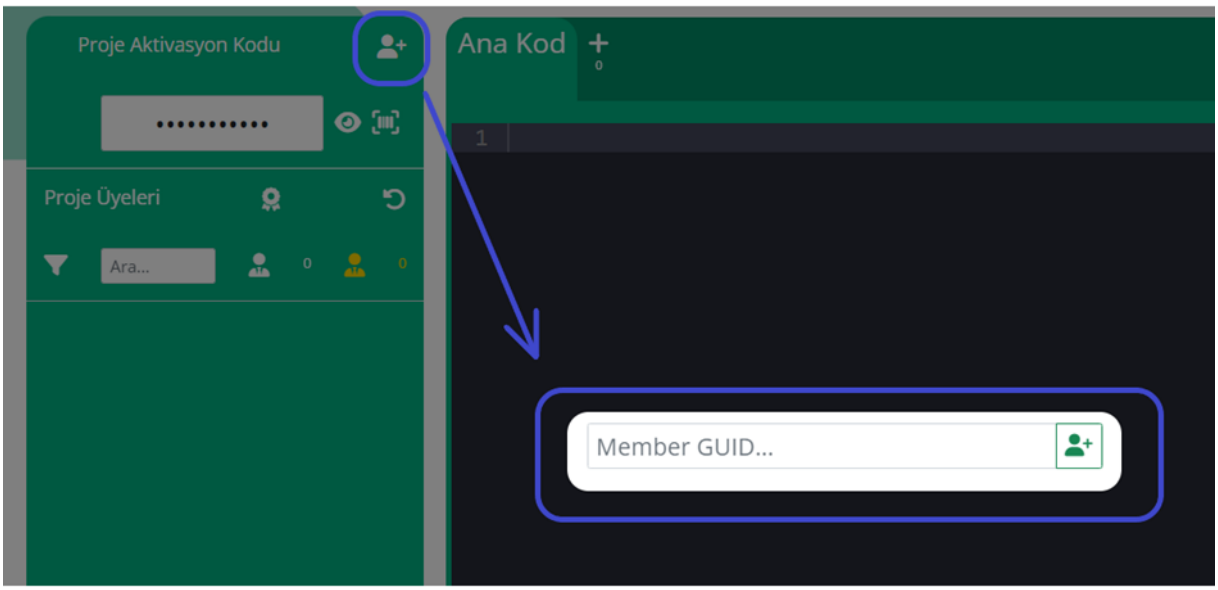
Eğer yeni bir hesap ise sadece geliştirici ortamından projeye dahil olunabilir.



Resim 8

1.4.2. Projeye Geliştirici Ortamından Dâhil Olmak

Geliştirici ortamında projeye dahil olma yöntemi seçildiğinde, aşağıda bulunan "Proje Aktivasyon Kodu" kısmında + (artı) butonuna tıklanması gerekmektedir. Ekrana "Member GUID" bilgisinin girilmesi için bir mesaj kutusu gelecektir.

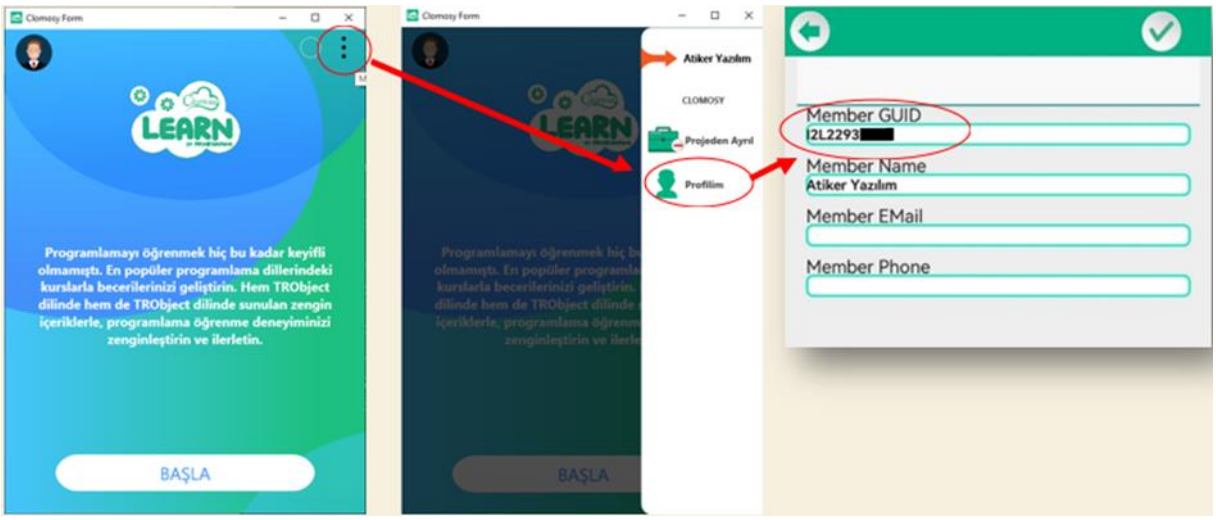


Resim 9

Girilecek olan GUID'e erişebilmek için mobil uygulamada açtığınız hesabınıza giriş yapmalısınız. Uygulama ekranının sağ üst köşesinde bulunan üç noktaya tıklayarak 'Profilim' seçeneğinden 'Member GUID'e ulaşabilirsiniz.

Mobil uygulamada yeni bir hesap oluşturulduğunda, ekranda "Clomosy Learn" uygulaması açılacaktır. Ekranın sağ üst köşesinde bulunan üç noktaya tıklayarak 'Profilim' seçeneğinden 'Member GUID'i öğrenebilirsiniz.

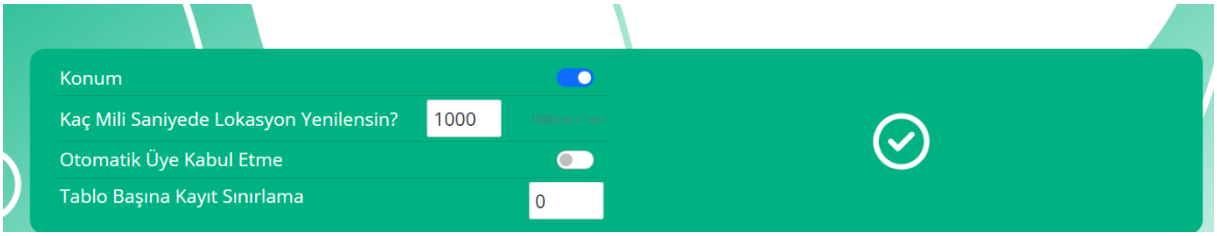
Ulaşılan Member GUID'i, yukarıda görselde (Resim 9) yer alan geliştirici tarafındaki mesaj kutusuna girerek istediğiniz kullanıcıyı, proje geliştirici ortamından projenize dahil edebilirsiniz.



Resim 10

1.4.3. Projeye Otomatik Dâhil Olmak

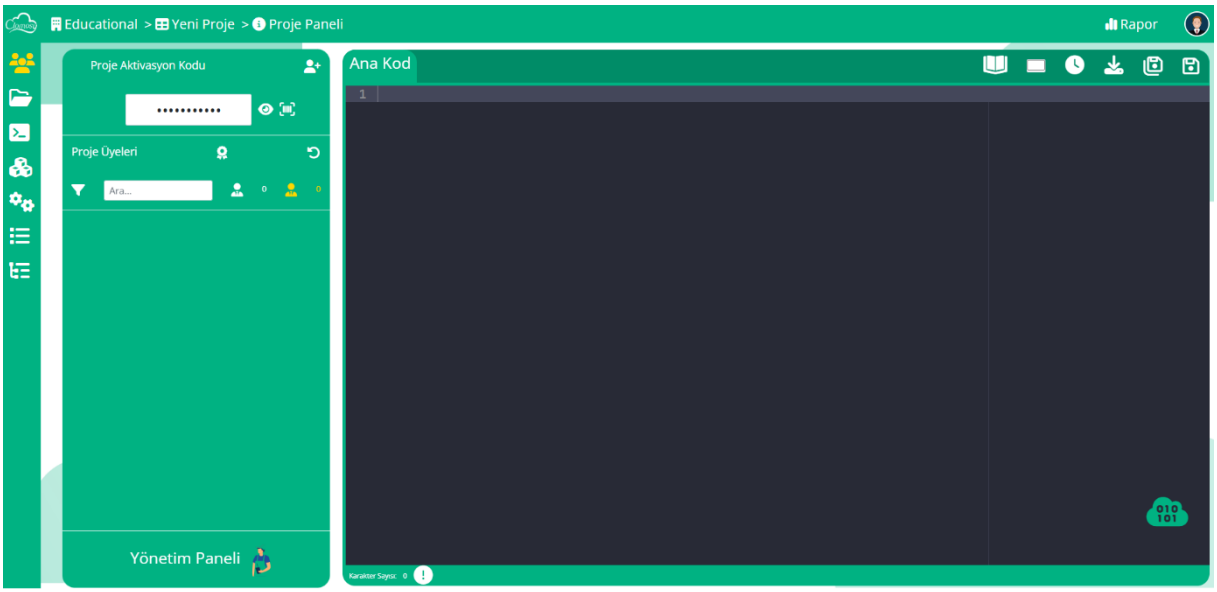
Bu yöntemde, geliştirici hesabına giriş yaparak Yönetim Panelinde yer alan "Parametreler" sayfasında "Otomatik Üye Kabul Etme" butonunu etkinleştirmek yeterli olacaktır. Ardından "Projeye Kullanıcı Ekranından Dahil Olma" yöntemi izlenmektedir. Bu yöntemde tek fark; kullanıcı projeye otomatik kaydolur. Geliştirici ekranında bulunan proje üyelerinin onaylanmasına gerek kalmaz.



Resim 11

1.5. Kod Editörü

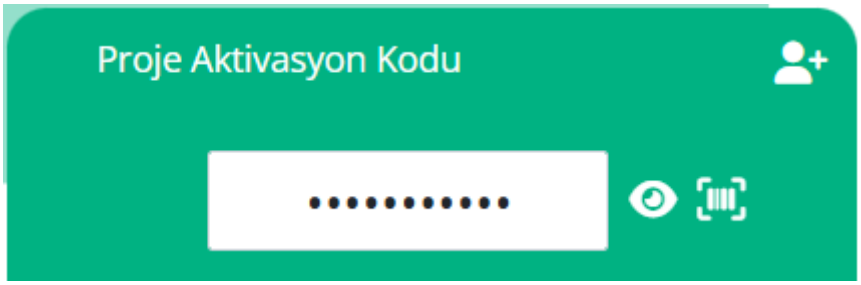
Clomosy üzerinde uygulama geliştirebileceğiniz ekran, aşağıdaki görselde yer almaktadır. Bu ekranın kullanımını inceleyelim.



Resim 12

1.5.1. Proje Aktivasyon Kodu

Geliştiriciler yeni proje oluşturduğunda, her projeye ait özgün bir aktivasyon kodu üretilmektedir. Aktivasyon kodu örneği aşağıda bulunan görselde gösterilmektedir. Aktivasyon kodu güvenlik için gizli tutulmaktadır. Yanında bulunan göz butonuna tıklandığında koda erişim sağlanmaktadır. Kullanıcılar aktivasyon kodu ile projeye dâhil olarak kullanmaya başlayabilir.



Resim 13

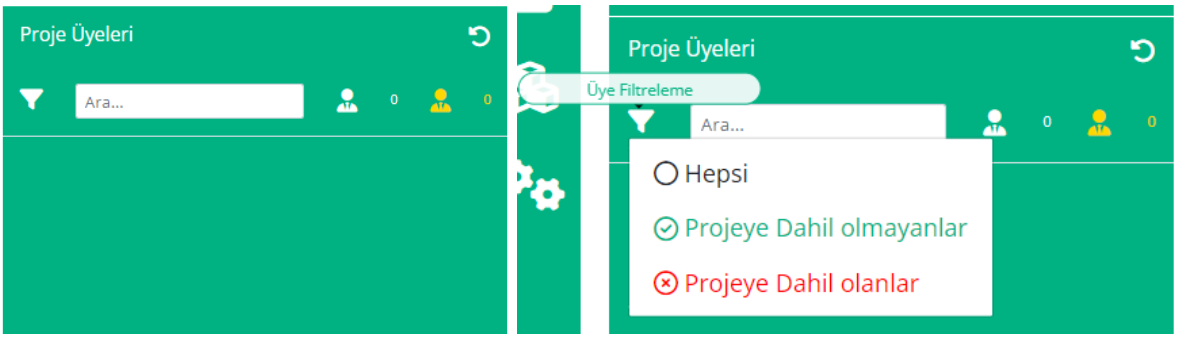
1.5.2. Proje Üyeleri Bölümü

Kullanıcılar tarafından projeye dâhil olma isteği gönderildiğinde liste güncellenmelidir. Güncelleme işlemi, sağ üst kısımda bulunan yenile butonu ile gerçekleştirilmelidir. Üye sayısı, arama kutucuğunun sağ tarafında yer almaktadır.

Üye listesini filtrelemek için sağ görselde (Resim 14) göründüğü gibi *Üye Filtreleme* butonuna tıklamak yeterlidir. Bu özellik ile üyelerin tamamını, projeye dâhil olanları ve olmayanları seçerek filtreleme yapılmaktadır.

Üye listesinde çok sayıda üye bulunduğunda, arama çubuğu kullanılarak kolayca arama gerçekleştirilebilir.

Clomosy Platformunun geliştiricilere özel sunmuş olduğu bir diğer özellik ise listelenen üyelerin GUID bilgisi kolaylıkla alınır. Projeye kayıtlı üye isminin üstüne tıklanıp açılan penceren GUID kopyalanarak proje içerisinde istenilen alana yapıştırılabilir.

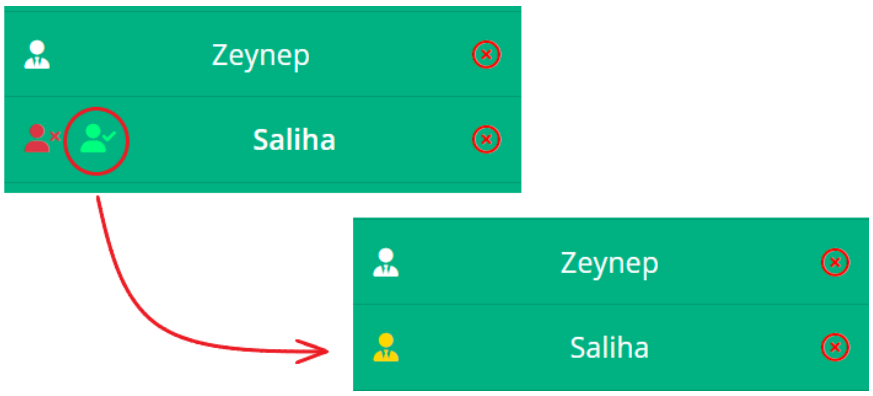


Resim 14

Bunun dışında başka bir özellik ise projeye kayıtlı olan üyeler arasında iki tip kullanıcı durumunun mevcut olmasıdır. İki tip kullanıcıların biri yönetici yetkisi verilen kullanıcı, diğeri normal kullanıcı olarak geçmektedir.

Yönetici yetkisine sahip olan kullanıcılar, sistem üzerinde ayrıcalıklı yetkilere sahip olması sağlanabilir.

Normal kullanıcı ise yönetici ayrıcalıklarına sahip olmayan, sınırlı yetkilere sahip normal bir kullanıcıyı ifade eder.



Resim 15

Yönetici yetkisi verilmesi için kullanıcın projeye dahil olması gerekmektedir. Ardından yukarıda görüldüğü gibi üyeler listesinde yönetici yapılmak istenen kullanıcın yanında bulunan profil butonuna tıklanır. Ardından yeşil tik ile gösterilen profil butonuna tıklanır ve kullanıcıya yönetici yetkisi verilir. Yönetici yetkisi verilen kullanıcının yanında artık sarı renkte bir profil butonu görülmektedir.

Yetkisi alınmak istendiğinde aynı işlem tekrar yapılır.

Bu yetkinin uygulama içerisinde kullanımı için Kullanıcı Verilerine Erişim bölümünden erişebilirsiniz.

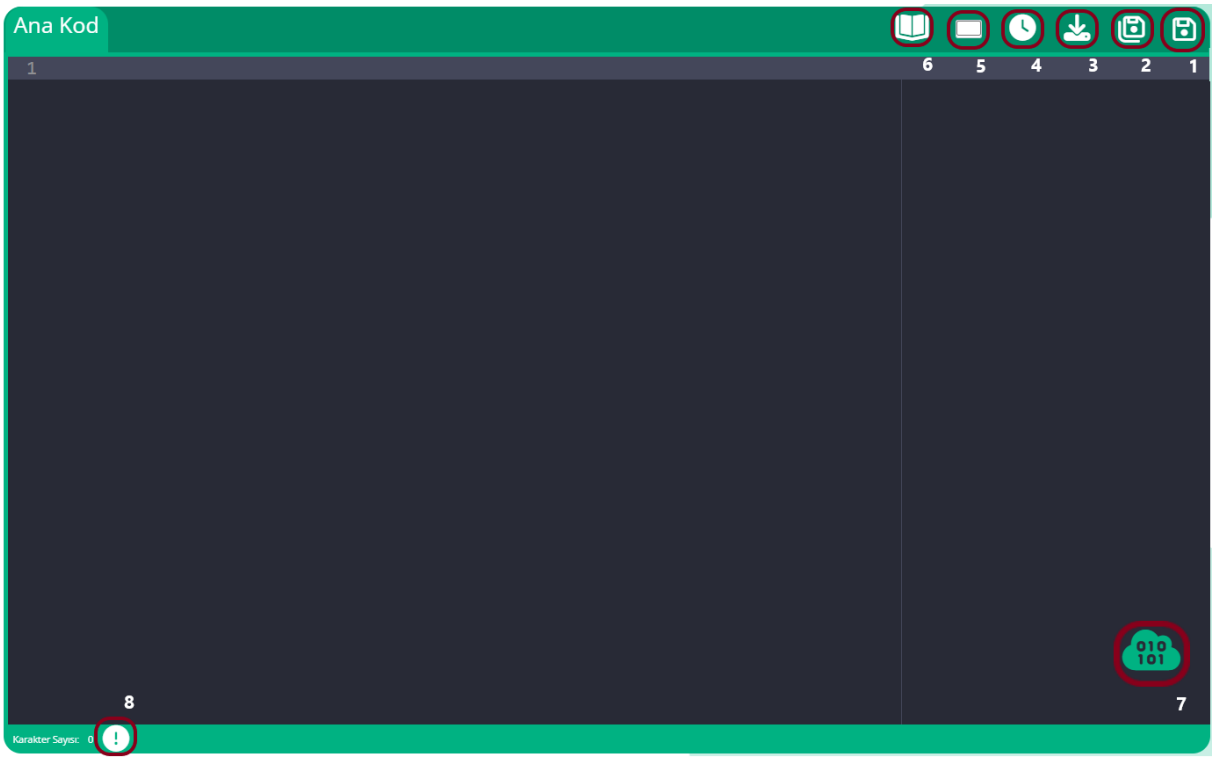
1.5.3. Yönetim Paneli

Clomosy'nin geliştiricilere sunmuş olduğu şablonlar (templates) bulunmaktadır. Bu şablonlar geliştiricilere hazır tablolar sunmaktadır. "Yönetim Paneli" butonundan bu şablonlara erişim sağlanmaktadır.

Yönetim paneli kullanımı, üyelik özelliğine göre değişmektedir. Premium hesap kullanıcısı, sayfa içerisindeki butonlara erişim sağlayarak tabloları kullanabilir. Fremium hesap kullanıcısı ise sadece "Aktif kodlamalar ve Parametreler" sayfasına erişim sağlamaktadır.

1.5.4. IDE Ayarları

Clomosy üzerinde proje geliştirmek için kullanılan kod geliştirme ortamı aşağıdaki görselde gösterilmektedir. Geliştiriciler tarafından birden fazla ayar kontrol edilebilmektedir. Aşağıdaki görselde, kontroller numaralandırma kullanılarak açıklanacaktır.



Resim 16

(1) : Ana Kod sayfasında yazılan kodları kaydetmek için kullanılan butondur.

(2) : Birden fazla sayfa (Unit) oluşturulduğunda hepsi aynı anda kaydedilmek isteniyor ise kullanılan butondur.

(3) : Ana Kod içerisinde yazılan kodların indirilmesi istenirse kullanılan butondur.

(4) : Clomosy'de kod yazarken kaydetmeden çıkma veya sayfayı yanlışlıkla yenileme durumunda, yazılan kodlar kaybolabilir. Bu durumu önlemek için otomatik kaydetme süresi ayarlanmaktadır.

(5) : Renk paleti kullanılarak ekrana tıklandığında bir noktanın renk kodu elde edilir veya renk paletinden seçilen bir rengin kodu Hexadecimal, HSL veya RGB formatında öğrenilir.

(6) : Proje geliştirirken dokümantasyon desteğine ihtiyaç duyulduğunda bu butona tıklayarak Clomosy doküman sitesine gidilebilir.

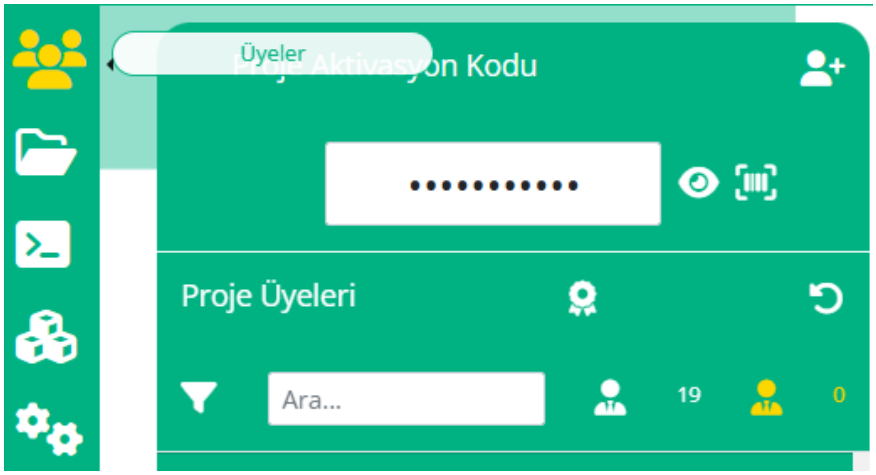
(7) : Nesne yardım aracıdır. Geliştirici bir nesne kullanmak istediğinde, o nesneye ait alanlar, özellikler ve metodlara bu buton üzerinden erişebilir. Butona tıklayıp aranan nesne yazıldığında erişebilir.

(8) : Yazılan kodu, kaydedip çalıştırdıktan sonra bir hata alındığında exe ve mobil cihaz üzerinde de görülmektedir. Hata mesajının ardından alınan hatayı görmek için *Kod Son Hatası* butonu bulunmaktadır. Bu buton son hatayı göstermektedir.

1.5.5. Menü Araç Çubuğu

Geliştirme ortamında bir dizi ayarı kolayca yönetebilmek için kullanılan bir araç çubuğu bulunmaktadır. Bu araç çubuğunda aşağıdaki özelliklere erişim mevcuttur.

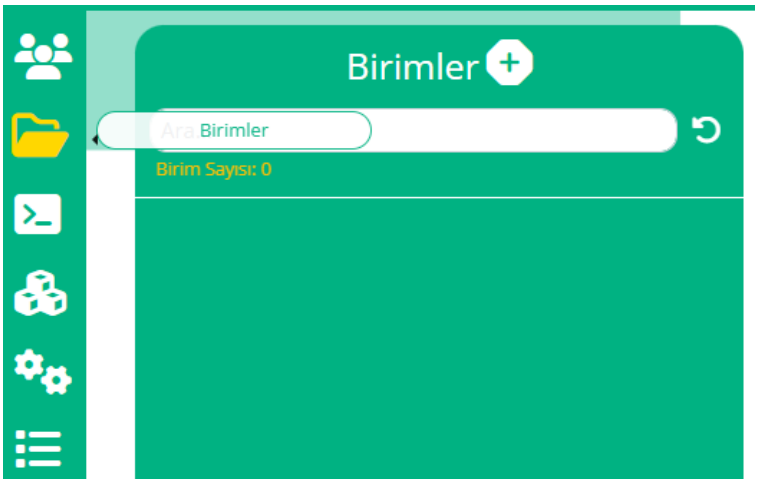
Üyeler listesine erişim sağlamak için kullanılan menü çubuğunda en üst kısımda buton yer almaktadır. Bu buton sayesinde projeye üye ekleme, çıkarma ve proje aktivasyon kodunun kontrolü sağlanmaktadır.



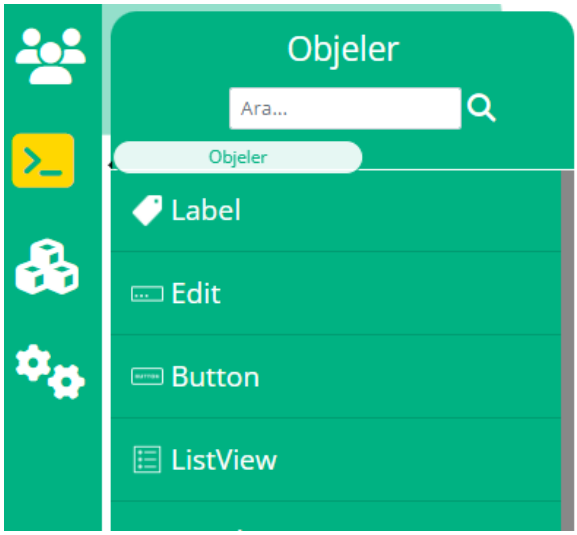
Resim 17

Birim oluşturabilmek için aşağıdaki gibi kod editöründe, sol menüde bulunan dosya ikonuna tıklanmalıdır. Bu kısımda "Birimler" bölümü açılır. Birim oluşturabilmek için artı (+) simgesine tıklandığında birim ekleme kutucuğu ekrana gelir.

Açılan "Birim Ekle" kutucuğunda 'Birim Adı' ve 'Birim Dili' girilebilecek alanlar yer alır. Bu şekilde yeni bir birim oluşturulabilir.

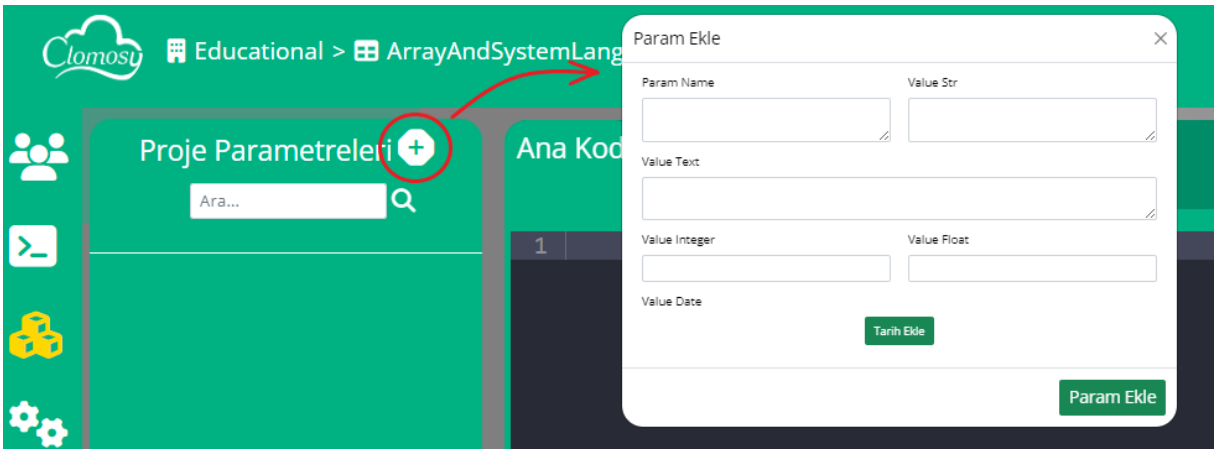


Clomosy üzerindeki nesnelere erişim, *Objeler* butonu sayesinde gerçekleştirilmektedir. Burada arama butonu ile bileşen aranabilir. Bileşenlerin kullanımını öğrenmek için kod editöründe boş bir alana imleci getirip bir bileşene tıkladığınızda, ilgili kodlar otomatik olarak imlecin bulunduğu alana yapıştırılmaktadır.



Resim 18

Kullanıcı Parametreleri butonu ile global bir değişken oluşturulmakta ve oluşturulan değişken bu projede istenilen unit üzerinde kullanılmaktadır. Aşağıdaki görselde, sekme üzerinde yer alan artı (+) butonuna tıklandığında "Param Ekle" kutusu açılmaktadır. Burada yeni parametre ekleme işlemi gerçekleştirilmektedir.



Resim 19

Yeni bir parametre eklemek için parametre adını 'Param Name' kutusuna yazmanız gerekmektedir. Yeni açılan parametrenin birden fazla veri depolama alanı bulunmaktadır. Bu alanlar; string veri için *Value Str*, integer veri için *Value Integer*, uzun metinler için *Value Text*, float veriler için *Value Float* ve tarih kaydı için *Value Date* alanına veriler eklenmektedir.

Bir örnek ile kullanımı nasıl yapılır gösterelim.

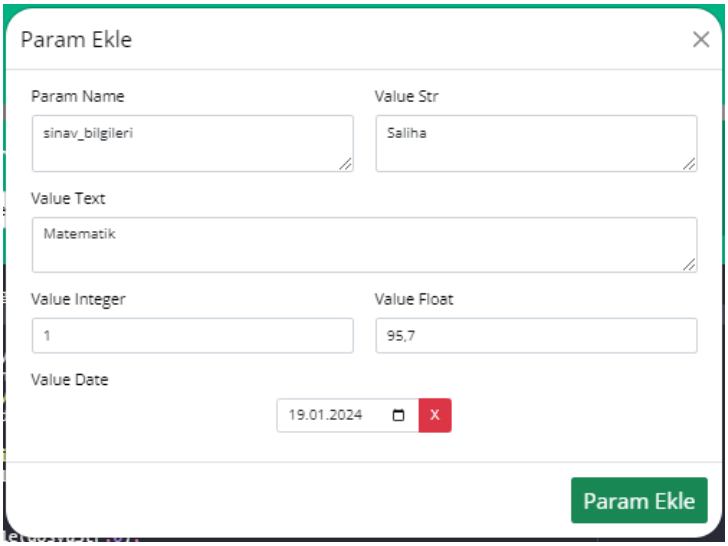
Bir öğrencinin sınav bilgisini tutan parametre ekleyelim. Bu parametre ismi öğrencinin sınav bilgisini kapsadığı için *Param Name* alanına sinav_bilgileri adında tanımlama yapılır. Öğrenci sınav bilgisi için;

Value Str: Adını

Value Text: Hangi dersin sınavı olduğu

Value Integer: Kaçıncı sınava girdiği

Value date: Sınav tarihi bilgisini girelim.



Resim 20

Bilgiler girildikten sonra verileri alabilmek için Clomosy alt yapısında bulunan *GetProjectUserDefParam* fonksiyonu kullanılmaktadır.

```
Clomosy.GetProjectUserDefParam(Param_Name).FieldByName('Field_Name').AsString;
```

Bu fonksiyonu örnek için kullanalım. Örneğimizde bulunan öğrenci adını almak için *Param Name* ve öğrenci adının bulunduğu alan adı (*Value_Str*) yazılmalıdır ve bu string veri türünde olduğu için string bir değişkene atılmalıdır.

```
Clomosy.GetProjectUserDefParam(sinav_bilgileri).FieldByName('Value_Str').AsString;
```

Örnek;

```
var
  ogrenciAdi,dersAdi,sinavTuru,sinavTarihi : String;

{
  ogrenciAdi =
Clomosy.GetProjectUserDefParam('sinav_bilgileri').FieldByName('Value_Str').AsString;
  dersAdi =
Clomosy.GetProjectUserDefParam('sinav_bilgileri').FieldByName('Value_Text').AsString;
  sinavTuru =
Clomosy.GetProjectUserDefParam('sinav_bilgileri').FieldByName('Value_Integer').AsString;
  sinavTarihi =
Clomosy.GetProjectUserDefParam('sinav_bilgileri').FieldByName('Value_Date').AsString;

  ShowMessage(ogrenciAdi+' '+dersAdi+' '+sinavTuru+' '+sinavTarihi);
}
```

Menü çubuğunda *IDE Ayarları* butonu, kod temasının değişimi, satır numaralarının gösterimi, dosya içe/dışa aktarma işlemleri gibi ayarlar yapılmaktadır.

Resim 21

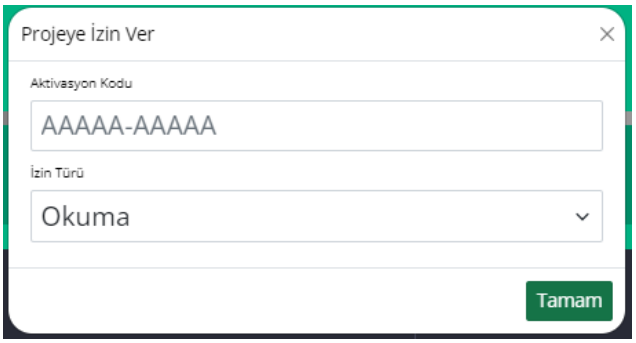
1.5.6. Proje İzinleri

Projenin bir başka kullanıcı tarafından görüntülenmesi için verilen okuma izni mevcuttur. Okuma izni sadece premium hesaplar için geçerlidir. Bir freemium hesabına sahip kullanıcı, başka bir kullanıcının projesini görüntüleme iznine sahip değildir.

Premium hesaplarda Projeye "Erişim İzni Verilmiş Listesi" ve "Erişim İzni Alınmış Listesi" butonu mevcut iken freemium hesapta sadece "Erişim İzni Verilmiş Listesi" butonu bulunmaktadır.

Resim 22

"Erişim İzni Verilmiş Listesi" butonuna tıklandığında menüde açılan + (artı) butonu aşağıdaki gibi bir bilgilendirme kutusu çıkarmaktadır. Bu kısma okuma izni verilecek projenin (Premium hesap projeleri) aktivasyon kodu girilerek 'Tamam' butonuna tıklandığında, projeye okuma izni verilmektedir.



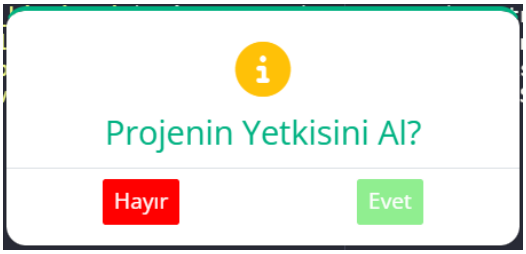
Resim 23

Projeye okuma izni verildiği anda listede okuma izni verilen projenin adı yer almaktadır.

Örneğin, Zeynep Hayal adlı geliştirici, Emin Demir adlı premium geliştiricinin “Araç Kiralama” projesine okuma izni verdiğinde aşağıdaki gibi sayfada okuma izni veren geliştiricinin adı (Zeynep Hayal) ve okuma izni verilen geliştiricinin proje adı (Araç Kiralama) yer almaktadır.

Resim 24

Geliştirici, projeden yetkilendirme iznini kaldırmak istediğinde proje adının yanında bulunan X (çarpı) butonuna tıklanmalıdır. Aşağıda görseldeki gibi uyarı mesajı geldiğinde 'evet' butonuna tıklayarak proje yetkisi kaldırılmaktadır.

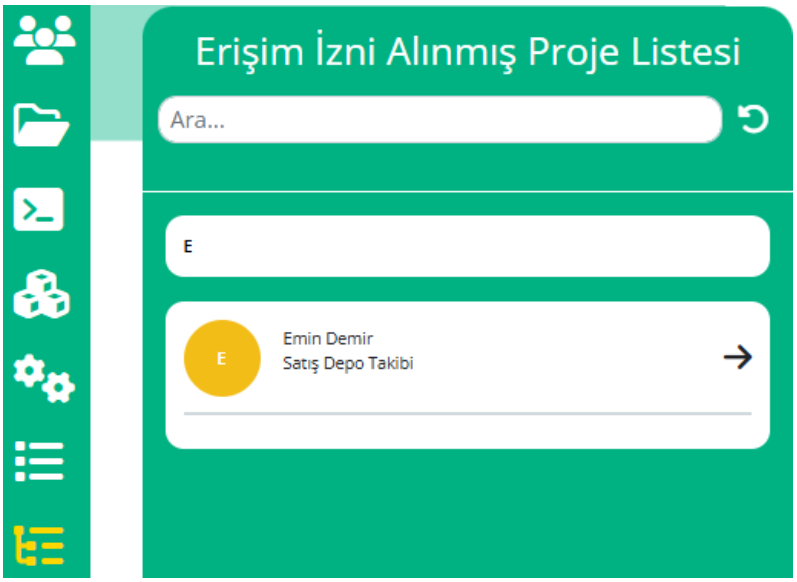


Resim 25

Premium hesaba sahip bir geliştirici bir başka geliştiricinin projesini okumak istiyorsa Geliştirici bir başka geliştiriciye proje okuma izni vermek istiyorsa öncelikle o geliştiricinin premium hesaba sahip olduğunu bilmesi gerekmektedir. Ardından geliştirici karşı tarafa proje aktivasyon kodunu vermelidir. Karşı taraf onayladıysa artık projeye okuma izni alınmıştır.

Premium hesapta izin alınmış projelerin kodlarını okuyabilmek için “Erişim İzni Alınmış Listesi” butonuna tıklanır. Bu butona tıklandığında izin alınmış projeler listesi gelmektedir. Hangi proje okunacaksa listeden o projeye tıklanmalıdır. Tıklamadan sonra geliştirme ortamına tıklanan projenin kodları gelir.

Örneğin, Zeynep Hayal adlı premium geliştirici, Emin Demir adlı geliştiricinin “Satış Depo Takibi” projesine okuma izni bulunmaktadır. Bu Zeynep Hayal adlı geliştiricinin “Erişim İzni Alınmış Proje Listesi” sayfasında aşağıdaki gibi görünür.



"Satış Depo Takibi" proje adının yanında bulunan ok butonuna tıklandığında Zeynep geliştiricisinin önüne Emir Demir geliştiricisinin "Satış Depo Takip" projesi kodları gelmektedir. Zeynep bu projeyi sadece okuyabilir. Okumadan çıkmak için aşağıda bulunan Ayrıl butonuna tıklanır ve Zeynep kendi proje ekranına geri gelmektedir.

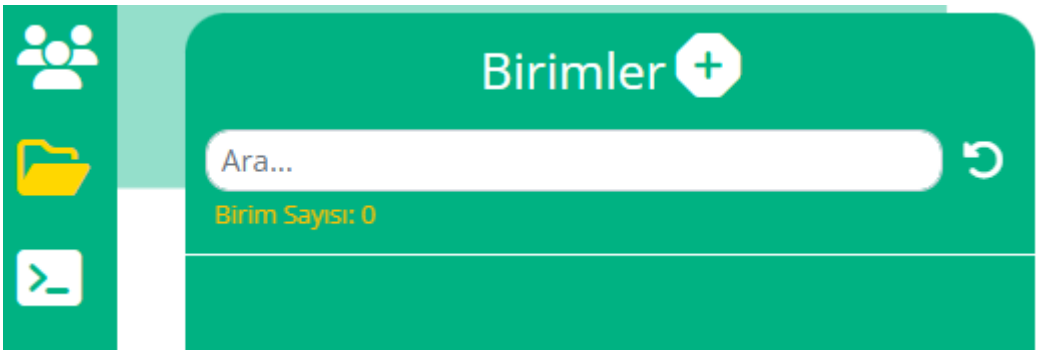
Resim 26

1.6.Unit Kullanımı

Bir projede, proje dosyasına ek olarak bir form dosyası ve bir kaynak kod dosyası oluşturulmaktadır. Bu kaynak kod dosyalarına, unit (birim) adı verilir ve dosya uzantısı, ".tro" şeklinde belirlenir. Bir projede birden fazla unit bulunabilir.

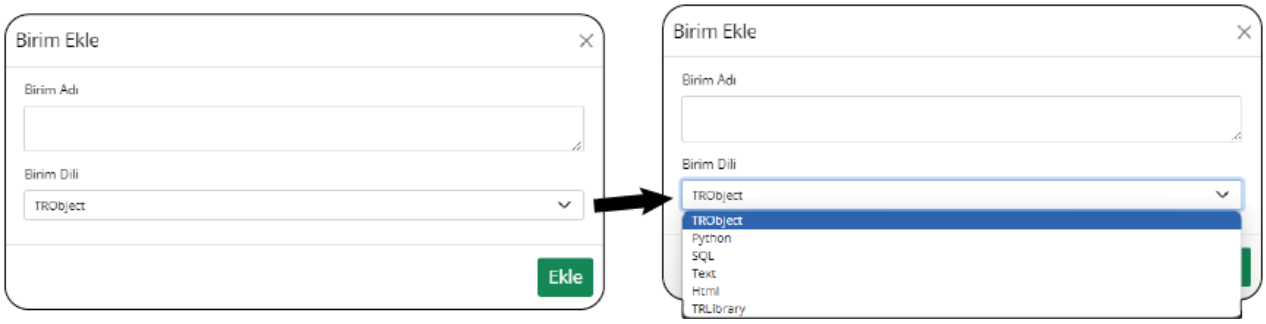
Clomosy'de *Unit*, birimlerin kodu organize eden ve parçalara ayıran bir yapıdır. Her *Unit* belirli işlevi yerine getiren bir kod paketidir. Unit, Clomosy projelerindeki birim dosyalarını ve farklı kod modüllerini temsil eder. Birleştirildiğinde kapsamlı bir uygulama oluşturmaya yardımcı olur. Unit; ayrı birimlerin işlevlerini düzenlemeye yardımcı olarak kodu daha okunabilir, yönetebilir ve bakımı kolay hale getirmektedir. Clomosy geliştirme ortamında bir *Unit* nasıl oluşturulur inceleyelim.

Geliştirme ortamında menüde bulunan "Birimler" butonuna tıklandığında birimler sayfası açılmaktadır.



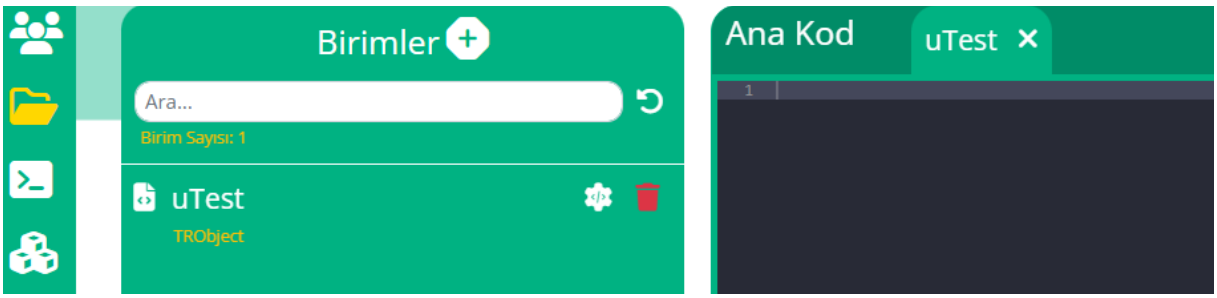
Resim 27

Burada bulunan artı (+) butonuna tıklandığında 'Birim Ekle' penceresi (Resim 28) ekrana gelmektedir. Birim adı verildikten sonra birim dili olarak TRObject seçilmektedir.



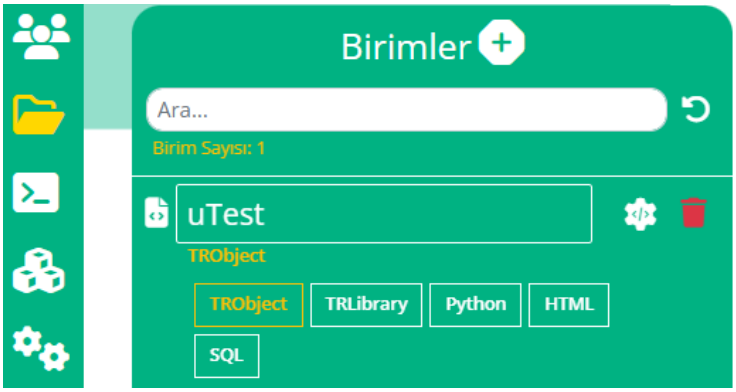
Resim 28

Unit ismi ve dil seçildikten sonra 'Ekle' butonuna tıklanarak oluşturulan Unit, 'Birimler' listesinde görüntülenir. Bu birime tıklandığında, sekme üzerinde ilgili birim açılır.



Resim 29

Birim adını değiştirmek için Birimler listesinde adı değiştirilmek istenen birim adının yanında bulunan ayarlama butonuna tıklanır. Birim adı üzerinde değişiklik yapılarak tekrar kapatılabilir.



Birimi silmek için aynı ekranda bulunan çöp kutusu simgesine tıklanmalıdır. Silinmek istenen birim, sekmede açık ise silme işlemi gerçekleştirilemez. Birimi silebilmek için öncelikle bu birim sekmede kapatılmalıdır.

CLomosy platformundan oluşturulan birimleri çağırma yöntemleri bulunmaktadır. Bunlar;

- RunUnit
- TclUnit

RunUnit

Clomosy kütüphanesinin sunmuş olduğu 'RunUnit' fonksiyonu, oluşturulan birimi çağırmak için kullanılmaktadır. Bu standart birim çağırma yöntemidir. Birim adını içeren bir parametre almaktadır. Bu parametre birim adını içermelidir.

```
Clomosy.RunUnit('uTest');
```

Bir örnek üzerinde görelim.

Örneğin; 'uMesajVer' adında birim (unit) oluşturulur. Bu birim içerisinde *ShowMessage* ile mesaj verilebilir. Ardından 'Ana Kod' ekranında 'RunUnit' ile çağırılır.

Ana Kod içerisine yazılan kod aşağıdaki gibidir.

```
ShowMessage('Ana Kod içerisindesin ve mesajVer birimine yönlendirileceksin.');
Clomosy.RunUnit('mesajVer');
```

uMesajVer birimi içerisine yazılan kod aşağıdaki gibidir.

```
ShowMessage('mesajVer birimi içerisindesin.');
```

TclUnit

Clomosy üzerinde standart birim çağırma yöntemi dışında özel birim çağırma yöntemi de bulunmaktadır. Bu yöntemin *RunUnit* yönteminden farkı, hangi birim üzerinden çağrıldıysa o birim içerisindeki nesnelere erişim sağlamasıdır.

Bu yöntem; TclUnit sınıfına ait bir nesnenin oluşturulması ile gerçekleştirilmektedir. Nesnenin kullanılması için *TclUnit* sınıfına ait nesne tanımlanır. Tanımlama işleminden sonra nesne *Create* ile oluşturulur.

Ana kod üzerindeki bir yapının başka bir unit üzerinde kontrollerinin sağlanması için kullanılan bir yöntemdir. Bu yöntemin kullanılması için ana form olarak kullanılacak birimin içerisinde yapılması gereken talimatlar bulunmaktadır.

UnitName özelliği ile tanımlanan *TclUnit* nesnesine gidilecek olan unit ismi atanmalıdır.

CallerForm ile gidilecek olan birimin hangi formdan çağrıldığını belirlemek için referans belirlenmelidir.

Tanımlanan *TclUnit* nesnesi *Run* ile çalıştırılmalıdır. Kullanımı aşağıdaki gibidir.

```
mainForm.clShow;
Unit2.UnitName = 'uYonlendirme';
Unit2.CallerForm = mainForm;
Unit2.Run;
```

Yukarıdaki kullanımda ana form nesnesi *mainForm* isminde tanımlanmıştır. Burada *uYonlendirme* birimine gidilmektedir. *mainForm* formunu referans almaktadır. Son olarak ana form içerisinde tanımlanan *TclUnit* nesnesi çalıştırılmaktadır.

Bir örnek ile pekiştirelim.

Örnekte bir tane *uYonlendirme* birimi (Unit) oluşturularak *anaForm* içerisinde *lblFormBilgi* nesnesine oluşturulan birim (Unit) tarafından değişiklik yapılacaktır. Bunun için *anaForm* içerisinde *TclUnit* nesnesi oluşturulmuş ve *CallerForm* özelliği *anaForm* olarak atanmıştır.

anaForm kodu:

```
var
  anaForm : TclStyleForm;
  btnUnit : TclButton;
  Unit1 :TclUnit;
  lblFormBilgi : tclLabel;

void BtnUYonlendirmeGit;
{
  anaForm.clShow;
```

```
  Unit1.UnitName = 'uYonlendirme';
  Unit1.CallerForm = anaForm;
  Unit1.Run;
}
{
  anaForm = TclStyleForm.Create(Self);
  Unit1 = TclUnit.Create;


  lblFormBilgi = anaForm.AddNewLabel(anaForm,'lblFormBilgi','Beklenen metin...');
  lblFormBilgi.Align = alMostTop;

  btnUnit = anaForm.AddNewButton(anaForm,'btnUnit','Özelleştirilmiş Unit');
  btnUnit.Align = alTop;
  btnUnit.Margins.Top = 20;
  anaForm.AddNewEvent(btnUnit,tbeOnClick,'BtnUYonlendirmeGit');

  anaForm.Run;
}
```

Ardından *uYonlendirme* birimi içerisinde bir edit ve button nesnesi oluşturulmuştur. Edit içerisine yazılan ne ise *anaForm* içerisindeki *lblFormBilgi* nesnesinin *Text* özelliğine atama yapılmaktadır.

uYonlendirme kodu:

```
var
  uForm : TclStyleForm;
  btnBilgiVer :TclButton;
  edtBilgi :TclEdit;

void BtnRunUnitSampleClick;
{
  CallerForm.clShow;
  lblFormBilgi.Text = edtBilgi.Text;
}

{
  uForm = TclStyleForm.Create(Self);
  edtBilgi= TclEdit.Create(uForm);
  uForm.clGetChild(edtBilgi);
  edtBilgi.Text= 'Gerekli açıklamayı giriniz...';
  edtBilgi.Align = alMostTop;

  btnBilgiVer = uForm.AddNewButton(uForm,'btnBilgiVer','Bilgiyi gönder');
  btnBilgiVer.Align = alTop;
  btnBilgiVer.Margins.Top = 20;
  uForm.AddNewEvent(btnBilgiVer,tbeOnClick,'BtnRunUnitSampleClick');
  uForm.Run;
}
```

BÖLÜM 2. Kodlama

Kodlama, genellikle bilgisayar programlarının oluşturulması ve yazılması sürecini ifade etmektedir. Yazılım geliştirici, bir program ya da uygulamanın mantığına göre talimatlar içeren bir dizi kod yazarak bilgisayarın anlayabileceği bir formata çevirmektedir. Kodlama süreci; bir programlama dilini kullanarak, yazılım geliştirme projelerinin temellerini oluşturmaktır. Bu süreç; problemleri çözme, işlevsellik ekleme ve belirli görevleri gerçekleştirmek için bilgisayar tarafından yürütülecek talimatları tanımlama sürecidir. Yazılım dilinin kurallarına, yapısına ve söz dizimine hâkim olmak gerekmektedir. Aksi durumda birçok yazım hatası meydana gelir.

2.1. Değişkenler

Değişken, bir değeri saklayabilen ve bu değere bir isim verilen bellek alanıdır. Bir değişken için ayrılan belleğin boyutu, içine yazılacak değere bağlıdır. Teorik olarak, tüm değişkenler için yalnızca büyük bellek "yuvaları" tahsis etmek mümkün olabilir. Ancak, durum böyle olsaydı, kullanılabilir belleğimiz hızla tükenirdi.

Bir değişkene tam olarak ne kadar bellek ayrılacağını bilmek için veri türünü bilmemiz gerekir.

Clomosy'de değişkenler, çeşitli görev ve nedenlerle kullanılmaktadır. Değişkenlerin kullanılmadan önce tanımlanması gerekmektedir. “var” anahtar kelimesi, prosedür tanımının sonrasında ve anahtar sözcük başlamadan önce yer alan değişken tanımlarının bir bölümünü başlatmak için kullanılmaktadır. Değişken tanımının formatı şöyledir:

var

<variable name> : <data type>;

Değişken adı, aşağıdaki söz dizimine sahip bir karakter dizisidir:

- Harfler, rakamlar (0-9) veya alt çizgi içerebilir.
- Bir harfle başlamalıdır.

Değişken adı herhangi bir uzunlukta olabilir (karakter sayısı sınırlı değildir) ve büyük/küçük harfe duyarlı değildir. Bir değişkeni adlandırmayı düşünürken, geçici veya döngü sayacı değişkenleri dışında tek karakterli adlardan kaçınmanız gerekmektedir. Döngü sayacı değişkenleri, I ve J şeklinde adlandırılır. Tek karakterli değişken adlarına örnek olarak S (dize) ve R (yarıçap) kullanılır. Tek karakterli değişken adları her zaman büyük harfle yazılmalı ve daha anlamlı adlar tercih edilmelidir. 1 (bir) rakamıyla karıştırılmaması için değişken adı olarak l (küçük L) harfini kullanmamaya dikkat edilmelidir.

```
Var
  K : Integer;
{
  ….
}
```

Yukarıdaki örnekte, tam sayı türünde bir k değişkeni bildirildi.

Clomosy'de birçok veri türü vardır. Bazılarına bir göz atalım.

Tam sayı veri türleri : Integer, Cardinal, ShortInt, SmallInt, LongInt, Int64, Byte, Word. Gerçek sayılar çeşitli veri türleri ile temsil edilebilir. Bunlar: Real, Single, Double, Extended, Comp, Currency. Gerçek sayıların gösterimi, normalleştirilmiş bilimsel gösterim (üstel gösterim) olarak da bilinir.

Örneğin:

3,28·1017 1,4·10–9 -5,101·104

Clomosy'de bu sayılar şu şekilde kaydedilir:

3.28e+17 1.4e-09 -5.101e+4

328e15 0.14e-8 -5101e+1

0.328e+18 140e-11 -510100e-1

Veri Tipi	Bayt Boyutu	Değer Aralığı
ShortInt	1	-128 ile 127
Byte	1	0 ile 255
Char	1	0 ile 255
WideChar	2	0 ile 65,535
SmallInt	2	-32,768 ile 32,767
Word	2	0 ile 65,535
LongInt	4	-2,147,483,648 ile 2,147,483,647
Int64	8	-9,223,372,036,854,775,808 ile 9,223,372,036,854,775,807
Integer	4	LongInt ile aynıdır
Cardinal	4	0 ile 2,147,483,647
Single	4	1.5 ¥ 10-45 ile 3,4 ¥ 1038
Double	8	5.0 ¥ 10-324 ile 1.7 ¥ 10308
Real	8	5.0 ¥ 10-324 ile 1.7 ¥ 10308
Extended	10	3,4 ¥ 10-4932 ile 1.1 ¥ 104932
Comp	8	-9,223,372,036,854,775,808 ile 9,223,372,036,854,775,807
Currency	8	-922,337,203,685,477.5808 ile 922,337,203,685,477.5807
Boolean	1	Doğru veya Yanlış
Variant	16	Değişken

Resim 30

2.1.1. Değişken Tipleri

Temelde birkaç değişken tipi bulunmaktadır. Bunlar;

String Metin verilerini depolamak için kullanılan bir değişken türüdür.

Char Tek bir karakteri depolamak için kullanılan bir veri türüdür. Tek tırnak işareti (‘’) arasına alınarak ifade edilmektedir.

Integer Tam sayılar için kullanılan bir terimdir, pozitif, negatif veya sıfır olabilir.

Boolean Doğru (True) veya yanlış (False) olarak ifade edilebilen, mantıksal değerler alan değişken türüdür.

Real Ondalık sayılar olarak da ifade ettiğimiz, nokta ile ayrılmış sayılar için de kullanılabilen değişken türüdür.

2.1.2. Değişkenlerin Bildirim Yerleri ve Türleri

Bir TRObject programında değişkenler önceden bildirilmelidir. Değişkenin nerede bildirildiği oldukça önemlidir. Eğer bir değişken, programdaki birden fazla yerde kullanılacaksa genel (global) olarak; yalnızca tek bir yerde (fonksiyon veya prosedür) kullanılıyorsa yerel (local) olarak bildirilmesi uygundur.

Yerel (Local) Bildirim

Yerel (Local) değişkenler kullanıldığı fonksiyon veye prosedürün içerisinde bildirilir; yalnızca bildirildiği fonksiyon (prosedür) içerisinde tanınır ve kullanılabilir. Bildirildiği fonksiyon (prosedür) dışında tanınmaz. Yerel (Local) değişkenler, tanımlandığı alanda (fonksiyon veya prosedür) çalıştırıldığında bellekteki veri alanında yer kaplarlar. Örneğin aşağıdaki programda bulunan prosedürde değişkenler local olarak tanımlanmıştır;

```
function sayilariCarp(a, b: Integer): Integer;
var
  LocalDegisken: Integer; //fonksiyon içerisindeki yerel değişken
{
  LocalDegisken = 10;
  Result = a * b * LocalDegisken;
}

var
  anaDegisken: Integer;
{
  try
  // Ana program içinde bir yerel değişken
  anaDegisken = 5;
  ShowMessage('Sonuç: '+ IntToStr(sayilariCarp(anaDegisken, 3)));
  except
  ShowMessage('Exception Class: '+LastExceptionClassName+' Exception Message:
'+LastExceptionMessage);
  }
}
```

Genel (Global) Bildirim

Genel (global) değişkenler bütün fonksiyon ve prosedürlerin dışında bildirilir. Bir değişken bütün program boyunca sürekli olarak kullanılıyorsa, genel olarak bildirilmelidir. Böyle bir değişken yerel olarak bildirilirse her fonksiyona (prosedüre) gönderilmesi gerekir. Bu da fazla parametre aktarımına neden olur. Programcı, değişkenleri bildirmeden önce hangilerinin yerel ya da genel olacağını belirlemelidir. Örneğin; yerel değişken örneğinde verilen program, genel değişken kullanılarak tekrar yazılırsa, aşağıda görüleceği gibi parametre aktarımına gerek kalmamaktadır.

```
 var
 MainVariable,b: Integer; //global değişken

function MultiplyNumbers;
var
 LocalVariable: Integer; //fonksiyon içerisindeki yerel değişken
{
 LocalVariable = 10;
 Result = MainVariable * b * LocalVariable;
}

{
 try
   MainVariable = 5;
   b = 3;
   ShowMessage('Sonuç: '+ IntToStr(MultiplyNumbers));
 except
   ShowMessage('Exception Class: '+LastExceptionClassName+' Exception Message:
'+LastExceptionMessage);
 }
}
```

2.1.3. Değişkenlere Başlangıç Değeri Verilmesi

Herhangi bir biçimde bildirilen değişkenin başlangıç değeri rastgele bir değerdir. Bu da o anda değişkene ayrılan bellek gözündeki değerlerdir. Bazı uygulamalarda değişkenin bir başlangıç değerinin olması istenir. Böyle durumlarda değişken bildirildikten hemen sonra ana kod kısmında değeri verilebilir. Örneğin,

```
var
  sayi : Integer;
{
  sayi = 10;
}
```

2.2. Sabitler ve Sabit Değerler

Değişkenler, bellekte verinin saklandığı alanlara verilen sembolik isim olup, programın yürütülmesi süresince içerikleri değiştirilebilir. Sabitler de bellekte verinin saklandığı alanlara verilen sembolik isimdir; bildirilirken verilen değerler programın çalıştırılması süresince değiştirilemez.

2.2.1. Sabit Değerler

Sabit değerler, başlangıç değeri verilen değişken bildirimi gibi yapılır. Ancak veri tipinin önüne *const* anahtar sözcüğü eklenmelidir.

Değişkenler gibi sabitlerin de tipleri vardır. Anonim ve isimlendirilmiş olabilirler. Anonim sabitlerin örnekleri aşağıdaki gibidir:

100	Integer
2.7	Real
7.12457e-03	Real
True	Boolean
14.0	Real
-37	Integer
‘asderc’	String

İsimlendirilmiş sabitlere sahip olmak önemlidir. Sabitler, değerleriyle birlikte programın deklarasyon bölümünde tanımlanır.

İsimlendirilmiş sabit tanımlama söz dizimi şöyledir:

Const

<sabit adı> = <değer>;

Sabit isimlerini belirleyen kurallar değişken isimleri için olan kurallarla aynıdır. İsimlendirilmiş sabitlerin tipi, otomatik olarak değerinden türetilir.

İsimlendirilmiş sabitler genellikle okunabilirliği ve program bakımını artırmak için kullanılır aynı zamanda kritik parametreleri göstermek için de kullanılabilir. Bir örnek vererek daha anlamlı hale getirelim:

```
Var
  control : boolean;
const
  pi = 3.14;
{
  control = True;
  if (control)
  {
    ShowMessage(FloattoStr(pi));
  }
}
```

Dereceleri (DEGR) radyana (RAD) dönüştürmek istiyorsanız, dönüştürme formülü Clomosy' de aşağıdaki gibi görünecektir:

RAD = (DEGR ∗ 3.14) / 180;

2.2.2. String

Clomosy' de dizelerle ilgili bazı deneyimlerimiz oldu. Örneğin, başlık veya metin gibi özellikler yalnızca dize değerlerine sahip olabilir. Dizeler tam olarak nedir ve onlarla nasıl çalışabilirsiniz inceleyelim.

Bir dize, tek tırnak içindeki sembol dizisidir. Bir dize değişkeni bildirmek için *String* türünü kullanmanız gerekmektedir:

```
var
   s : String;
```

Bu tanım, programın sınırsız uzunlukta bir dizi değişkeni kullanabileceği anlamına gelmektedir.

Dizeler birleştirilebilir. Dize birleştirme artı işaretiyle gösterilir. Örneğin:

```
var
  s, st: String;
{
  s = 'Öğreniyoruz';
  st = 'Clomosy';
  s = st +' '+ s;
  ShowMessage(s);
}
```

Program çalıştığında ‘S’ değişkeninde “Clomosy Öğreniyoruz” karakter dizesini alacağız.

Bunun dışında dizeler karşılaştırılabilir. Karşılaştırma, sembollerin karşılaştırma ilişkisi kullanılarak (iç temsillerinin karşılaştırılması) gerçekleştirilir. Latin karakterlerin alfabetik sıralaması ve rakamların doğal sıralaması vardır. Burada 0 < 1 < … < 9.

Örnek:

‘AB’ > ‘AA’

‘A’ < ‘AB’

‘DC’ > ‘ABCDE’

‘ABCE’ > ‘ABCD’

Birkaç standart işlev ve yordam dizelerle çalıştırılmaktadır. Aşağıdaki tablo bunları özetlemektedir.

String Prosedürler		
İsim ve Parametreler	Parametre Tipleri	Anlamı
Delete(St, Pos, N)	St: string; Pos, N: integer;	Pos değişkeni pozisyonundan başlayarak N sembolünü silmek için, çoğu programlama dilinde string dilimleme kullanılabilir.
Insert(St1, St2, Pos)	St1, St2: string; Pos: integer;	St1'deki sembolleri, St2' deki Pos pozisyonundan başlayarak eklemek için, çoğu programlama dilinde string dilimleme ve string birleştirme kullanılabilir.
String Fonksiyonlar		
İsim ve Parametreler	Parametre Tipleri	Anlamı
Copy(St, Pos, N)	Result type: string; St: string; Pos, N: integer;	Sonuç, St stringinden Pos pozisyonundan başlayarak N sembolünün kopyasıdır.
Length(St)	Result type: integer; St: string;	St stringinin uzunluğunu (sembol sayısı) hesaplar.

Örnekler:

Delete prosedür:

St Değeri	İfade	İfade Çalıştırıldıktan Sonra St Değeri
‘abcdef’	Delete(St,4,2)	‘abcf’
‘Clomosy-Öğreniyorum’	Delete(St,1,7)	‘-Öğreniyorum’

Insert prosedür:

St1 Değeri	St2 Değeri	İfade	İfade Çalıştırıldıktan Sonraki Çıktı
‘Clomosy’	‘-Öğreniyorum’	Insert(St1,St2,1)	Clomosy-Öğreniyorum
‘Cl’	‘Kütüphanesi’	Insert(St1,St2,3)	KüCltüphanesi

Copy fonksiyonu:

St Değeri	İfade	İfade Çalıştırıldıktan Sonra St Değeri
‘abcdefg’	Str = Copy(St,2,3)	‘bcd’
‘abcdefg’	St = Copy(St,4,4)	‘defg’

Length fonksiyonu:

St Değeri	İfade	İfade Çalıştırıldıktan Sonra N'nin Değeri
‘abcdefg’	N = Length(St)	7
‘Clomosy Öğreniyorum’	N = Length(St)	19

2.3. Tip Dönüşümleri

Tip dönüşümleri, bir veri türünün başka bir veri türüne dönüştürülmesi işlemidir. Programlama dillerinde genellikle veri türleri arasında dönüşüm yapmak gerekebilir çünkü farklı işlemler veya fonksiyonlar farklı veri türlerini kullanabilir. Tip dönüşümleri, bir veri türünün başka bir veri türüne dönüştürülmesini sağlar ve bu şekilde uyumsuzlukları giderilir.

Örneğin, bir tamsayıyı ondalık bir sayıya dönüştürmek veya bir metni tam sayıya dönüştürmek gibi durumlar tip dönüşümlerine örnektir. Bu tür dönüşümler, programın gereksinimlerine ve mantığına bağlı olarak sık sık kullanılır.

StrToInt

Bir string veri türü, integer veri türüne dönüştürülmektedir.

```
var
  sayiStr : String;
  sayiInt : Integer;

{
   sayiStr = '27';
   sayiInt = StrToInt(sayiStr);
}
```

Örnekte string veri türünde bir değişken oluşturulmaktadır ve bu değişkene sayı değeri atanmaktadır. Ardından sayısal işlemler yapılabilmesi için integer veri türüne dönüştürülerek kullanılması sağlanmaktadır.

StrToFloat

Bir string veri türü, float veri türüne dönüştürülmektedir. StrToInt fonksiyonundaki örnek üzerindeki gibi aynı şekilde değer atamaları yaparak bu sefer float veri türündeki bir değişkene atama gerçekleştirelim.

```
var
  sayiStr : String;
  sayiFloat : Float;

{
   sayiStr = '27,45';
   sayiFloat = StrToFloat(sayiStr);
}
```

Dönüşümlerde float veri türünde tanımlanmış değişkene integer dönüşümü yapılmaya çalışılırsa hata ile karşılaşılır. Bunun için tip dönüşümlerine dikkat edilmelidir.

StrToIntDef

String bir veri türünü, integer veri türüne dönüştürmektedir. Dize tamsayıya dönüştürülemezse bu işlev, bir varsayılan değeri (default değeri) geri döndürür. Bu potansiyel olarak hatalı dönüşümleri önlemeye yardımcı olmaktadır.

Bu fonksiyon iki parametre şeklindedir. İlk parametre dönüşecek değer veya tipi, ikinci parametre ise herhangi bir hata durumunda varsayılan değerin ne olacağını belirtmektedir. Varsayılan değeri döndürebilen fonksiyonun sonu *Def* ile sonlandırılmaktadır.

```
var
  sayiStr : String;
  sayiInt : Integer;

{
   sayiStr = '27';
   sayiInt = StrToIntDef (sayiStr,0);
}
```

Örnekte *sayiStr* değişkeni içerisindeki değer, integer tipine dönüştürülebileceği için *sayiInt* değişkeni 27 değerini döndürecektir. Şimdi örneği değiştirelim ve *sayiStr* değişkenine integer türüne dönüştürülmeyecek bir değer verelim. Burada 'sayi değeri' veriyoruz. Bu değer integer türüne dönüştürülemeyeceği için default değer olan 0 değeri dönecektir.

```
var
  sayiStr : String;
  sayiInt : Integer;

{
   sayiStr = 'sayi değeri';
   sayiInt = StrToIntDef (sayiStr,0);
}
```

IntToStr

Integer veri türünü, string veri türüne dönüştürmektedir.

```
var
  sayiStr : String;
  sayiInt : Integer;

{
   sayiInt = 85;
   sayiStr = IntToStr(sayiInt);
}
```

Yazılım geliştirme sürecinde sıklıkla kullanılan diğer tip dönüşüm fonksiyonları ve işlevleri aşağıda gösterilmiştir.

Fonksiyon	Dönüşecek Veri Tipi	Dönüşen Veri Tipi
StrToInt	String	Integer
StrToFloat	String	Float
IntToStr	Integer	String
StrToDate	String	TclDate
StrToTime	String	TclTime
FloatToStr	Float	String
TimeToStr	TclDateTime	String

2.4. Operatörler

2.4.1. Atama Operatörü

Atama operatörü, programlama dilinde temel ve önemli bir talimattır. Atama sembolü TRObject dil yapısında sadece (=) eşittir sembolü ile gösterilmektedir.

Dâhili olarak çalışma şekli; önce atama operatörünün sağındaki ifadenin değerinin hesaplanması ve ardından elde edilen değerin atama operatörünün solundaki değişkene atanması şeklindedir.

<variable name> = <expression>;

2.4.2. Aritmetik Operatörler

Değişkenler ve sabitler üzerinde temel aritmetik işlemleri gerçekleştiren operatörlerdir. Bunlar;

Aritmetik Operatörler		
Oparatör	**Açıklama**	**Örnek**
+	Toplama	**x = 5 + 2;**
-	Çıkarma	**x = 10 - 7;**
*****	Çarpma	**x = 8 * 4;**
/	Real sayı bölümü	**x = 6 / 3;**
div	Tam sayı bölümü	**x = 10 div 7; (x = 1)**
mod	Bir sayının başka bir sayıya bölünmesi sonucunda kalan değeri verir.	**x = 10 mod 20; (x = 10)**
xor	İki sayının bit düzeyinde karşılaştırılması ile kullanılır. İki bit aynı ise sonuç 0, farklı ise sonuç 1 olur. Mantıksal işlemler ve şifreleme algoritmalarında kullanılır.	**x = 5 xor 3;** **//5: 0101, 3: 0011** **//x = 6 (sonuç 0110 olur)**
shl	Bir sayının bitlerini sola kaydırır ve kaydırılan her bit için sayının değeri 2 ile çarpılır.	**x = 3 shl 2;** **//3 : 0011** **//x = 1100 (yani 12 olacaktır)**
shr	Bir sayının bitlerini sağa kaydırır ve kaydırılan her bit için sayının değeri 2'ye bölünür.	**x = 8 shl 2;** **//8 : 1000** **//x = 0010 (yani 2 olacaktır)**

Resim 31

Örneğin:

13 div 4 = 3 13 mod 4 = 1

-13 div 4 = –3 -13 mod 4 = –1

İki Sayı Toplama

İlk olarak; üç adet Integer tipinde Sayi1, Sayi2 ve TplmDeger adında değişkenler tanımlanır.

```
Var
  Sayi1, Sayi2, TplmDeger : Integer;
```

Sayi1 ve *Sayi2* değerlerine sırasıyla değer ataması yapılır. Bu değişkenler, artı (+) operatörü ile toplanarak *TplmDeger* değişkeni içine yazdırılır. ShowMessage ile ekranda gösterilen sonuçta, Integer bir değer yer alacaktır. Bu değeri yazarken yanında String bir metin girilmek istendiğinde tamsayıları metne dönüştürme fonksiyonu *IntToStr* (integer) kullanılmalıdır.

```
{
  Sayi1 = 10;
  Sayi2 = 20;
  TplmDeger = Sayi1 + Sayi2;
  ShowMessage('Toplam deger: '+IntToStr(TplmDeger));
}
```

2.4.3. İlişkisel Operatörler

İlişkisel operatörler iki değeri karşılaştırmak için kullanılır. Karşılaştırmanın sonucu TRUE veya FALSE değerine sahiptir.

İlişkisel Operatörler		
Operatör	**Açıklama**	**Örnek**
= =	Eşittir	**(x = = 10)**
< >	Eşit değil	**(x < > 10)**
not	Değil ifadesi	**(not x == 10)**
<	Küçük ise	**(x < 10)**
>	Büyük ise	**(x > 10)**
<=	Küçük ve eşit ise	**(x <= 10)**
>=	Büyük ve eşit ise	**(x >= 10)**

Resim 32

Örnek:

```
var
        X: Real;
        Exist, Ok: Boolean;
{
        X = 2.5;
        Ok = X > 0;
        Exist = (not X == (16.0-2.1));
}
```

Bu programın sonucunda *Ok* değişkeni **TRUE** değerini, *Exist* değişkeni ise **FALSE** değerini depolamaktadır.

2.4.4. Mantıksal Operatörler

Mantıksal operatörler, mantıksal değerlerle birlikte kullanılır ve mantıksal bir değer de döndürürler. Aşağıdaki mantıksal operatörleri inceleyelim:

Mantıksal Operatörler		
Operatör	**Açıklama**	**Örnek**
&&	Mantıksal ve	**(x = 10) && (y = 2)**
||	Mantıksal veya	**(x = 10) || (y = 2)**

Resim 33

İfadenin değeri belirli bir sırayla hesaplanır. Operatör Önceliği Tablosu:

İfade Türü	Operatör
Parantez içindeki değerlendirme	( )
Fonksiyon değerlendirmesi	function
Tekli operatörler	not, unary '-'
Çarpma gibi operatörler	*, /, div, mod, and
Operatörler eklemek	+,-, or
İlişki operatörleri	=, <, >, <>, <=,>=

Resim 34

Aynı önceliğe sahip operatörler, ifadedeki sıralarına göre soldan sağa doğru değerlendirilmektedir. Örneğin; operatörlerin sırasını inceleyerek bir sonraki ifadenin değerini bulalım:

(a∗2>b) || not (c=7) && (d-1<=3), if a=2, b=4,

c=6, d=4.

(2∗2>4) || not (6=7) && (4–1<=3)

(4>4) || not (6=7) && (3<=3)

false || not false && true

false || true && true

false || true

True

Clomosy'de $4 < x \leq 18.3$ matematiksel ifadesi şu şekilde yazılmaktadır.

(x > –4) && (x <= 18.3)

2.5. Hata Yakalamak

Uygulama geliştirme sürecinin en önemli aşamalarından biri test sürecidir. Bu aşamada yapılan hata düzeltilebilir. Hataları en aza indirmenin, uygulamaya olan güveni artıracağı da göz önünde bulundurulmalıdır. Bu nedenle uygulamayı dağıtmadan önce en stabil durumda çalışılabilirliğini sağlamak amacıyla, kodlar bir kez daha kontrol edilmelidir. Tahmin edilebilir veya öngörülemeyen hataları bularak kullanıcıyı uyaracak nitelikte yeniden düzenlenmelidir. Hataya sebebiyet verecek etkenler arasında geliştirici, kullanıcı, network alt yapısı, işletim sistemi, donanımsal yapı gibi unsurlar sayılabilir. Hataya sebep olabilecek birçok ihtimali kodlarla engellemek mümkün olmaktadır.

Hata denetimi sonucunda kod akış yönü değiştirilebilir, kullanıcı uyarılabilir veya alternatif algoritmalara yönlendirilebilir.

2.5.1. Try-Except

Hata olasılığı yüksek olan kod satırları **try-except** bildirimleri arasına yazılır. Hata oluşması durumunda, **except** bloğundaki kod satırları çalışır. Hata olmaması durumunda **try-except** arasındaki kod satırları çalışır.

Try-except ifadesinin söz dizimi aşağıdaki gibi gösterilmektedir.

```
try
  //Hata yoksa bu bölümde bulunan kod satırları çalışır
except
  //hata olması durumda bu bölümde bulunan kod satırları çalışır
}
```

Try-except satırları arasına tek veya daha fazla kod satırları yazılabilir. Bu yöntemde herhangi bir hatada 'except' ifadesinden sonraki kod satırları işletildiği için hatanın sebebini bilmek zor olabilir. Bu nedenle birçok olasılık değerlendirilmeli ve ona göre bir iş akışı oluşturulmalıdır. Gerekirse her ihtimal için bir koşul belirtilerek kullanıcı yönlendirilmelidir.

Örneğin;

```
var
  sonuc : Integer;
{
  sonuc = StrToInt(testEdit.Text) div StrToInt(testEdit2.Text);
  ShowMessage(format('%s sayısının %s sayısına bölümünden %d elde edilmiştir.',
[testEdit.Text,testEdit2.Text,sonuc]));
}
```

Örnekte, bir sayı diğerine bölünmekte ve elde edilen sonuç kullanıcıya gösterilmektedir. Basit gibi görünen bu kod satırlarında hata olma ihtimali nedir?

Örneğin,

- Kullanıcı tarafından sayısal olmayan bir değer girilmiş olabilir.
- İkinci sayı sıfır olabilir.
- Herhangi bir sayı sıfıra bölünmeyeceğinden bir hata oluşacaktır.
- Geliştirici tarafından doğru tip dönüşümü yapılmamış olabilir.
- Yine geliştirici tarafından format fonksiyonunda doğru tip dönüşümü yapılmamış olabilir.

Bu durumda yukarıdaki ifadeyi aşağıdaki gibi yazmak doğru olacaktır.

```
var
sonuc : Integer;
{
  if ((testEdit.Text == '') || (testEdit2.Text == ''))
  {
    ShowMessage('Bölme işlemi için her iki alanada bir değer yazmak zorundasınız.');
    exit;
  }
  if ( IntToStr(testEdit2.Text) == 0)
  {
    ShowMessage('Sayı sıfıra bölünemez.');
    exit;
  }
  sonuc = StrToInt(testEdit.Text) div StrToInt(testEdit2.Text);
  ShowMessage(format('%s sayısının %s sayısına bölümünden %d elde edilmiştir.', [testEdit.Text,testEdit2.Text,sonuc]));
}
```

Bazen ihtimal dahi verilmeyen veya gözden kaçan birçok hata olabilir. İşte bu ve benzeri ihtimalleri en aza indirmek için try-except hata yakalama ifadeleri kullanılmaktadır.

Örneği aşağıdaki gibi değiştirelim.

```
var
sonuc : Integer;
{
  try
    sonuc = StrToInt(testEdit.Text) div StrToInt(testEdit2.Text);
ShowMessage(format('%s sayısının %s sayısına bölümünden %d elde edilmiştir.', [testEdit.Text,testEdit2.Text,sonuc]));
  except
    ShowMessage('Sebebini bilmediğim bir hata oluştu!');
  }
}
```

Try-except satırları arasında herhangi bir hata meydana geldiğinde except bloğu çalışacaktır. Try-except genel hata yakalayıcı olduğu için hatanın sebebini bilmek oldukça zordur. Bu nedenle olası bir hatanın ciddiyetine göre except bloğu içerisinde kullanıcı yönlendirilmeli veya alternatif bir durum belirlenmelidir.

Try-except yapısında hataya sebep olan durumu daha anlaşılır hale getirmek için sistem tarafından üretilen hata mesajlarını yakalamak mümkündür. Bunun için except bloğu içerisinde istisna sınıf adını alabilmek için 'LastExceptionClassName' ve istisna mesajını alabilmek için 'LastExceptionMessage' kullanılmaktadır. Kullanımı aşağıdaki gibidir.

```
Try
  …
except
  ShowMessage('Exception Class: '+LastExceptionClassName+' Exception Message: '+LastExceptionMessage);
}
```

2.5.2. Try-Finally

Herhangi bir hata oluşması durumunda veya hataya bağlı olmaksızın bazı kod parçalarının mutlaka işletmesi gerekiyorsa, ilgili kod satırları try-finally ifadesi ile kullanılır.

Try-finally ifadesinin söz dizimi aşağıdaki gibi gösterilmektedir.

```
try
  //Bu bölümde bulunan kod satırları çalışır veya bir hata durumunda çalışmayabilir.
finally
  //Bu bölümde bulunan kod satırları mutlaka çalışacaktır.
}
```

Oluşturulan bir sınıf veya tip işlem sonucunda yok edilmediğinde aşırı bellek tüketimine bağlı olarak birçok sorun meydana gelecektir. Bu durumda, oluşturma try ifadesinde, yok etme işlemi ise finally kod satırı sonrasında yazılır.

Aşağıdaki örnek uygulamayı inceleyelim.

```
var
  List: TclStringList;
{
  List = Clomosy.StringListNew;

  Try
    List.Add('Item 1');
    List.Add('Item 2');
    List.Add('Item 3');
  finally
     List.Free;
  }

}
```

Örnekte, TclStringList tipinde bir liste değişkeni oluşturulmuş ve listeye veriler eklenmiştir. TclStringList tipindeki değişkenin bellekte kapladığı yer, eklenen veri kadar sürekli artacaktır. Bu yüzden liste değişkenine ihtiyaç duyulmadığı anda yok edilmelidir. Örnek uygulamada, finally satırından sonra Free edilmektedir.

2.6. Mesaj Pencereleri

Kullanıcılara bir uyarı vermek, bilgilendirmek veya bir onay istemek için mesaj pencereleri kullanılır.

2.6.1. ShowMessage

Kullanıcıya bilgi vermek amacıyla kullanılan mesaj penceresidir. ‘Uygulama mesajı’ ve ‘tamam’ (ok) butonu olmak üzere iki ana kısımdan oluşmaktadır. Tamam butonunun dışında tıklama seçeneği yoktur. Mesaj penceresi ekranın ortasında görüntülenir.

Metodun temel yapısı aşağıda gösterilmektedir.

ShowMessage('Clomosy Learn!');

2.6.2. Ask

Kullanıcı onayı gereken durumlarda, iletişim kutusunu görüntülemek ve kullanıcıya soru sormak, seçenekler sunmak için kullanılır. Sorguda ‘Evet’ butonuna tıklanırsa fonksiyon “True” değerini, ‘Hayır’ butonuna tıklanırsa fonksiyon “False” değerini döndürmektedir. *Ask* yöntemi yalnızca Windows için desteklenir.

Genel kullanımı aşağıdaki gibidir.

```
if Clomosy.Ask(' Çıkış yapmak istediğinize emin misiniz?')
      //Sorguda ‘Evet’ butonuna tıklandığında çalışacak kod
else
      //Sorguda ‘Hayır’ butonuna tıklandığında çalışacak kod
```

2.6.3. AskAndCall

Ask fonksiyonuna benzer bir yapıya sahiptir. Kullanıcı onayı gereken durumlarda iletişim kutusunu görüntülemek için kullanılır. *AskAndCall* işlevi üç parametre alır. İlk parametre iletişim kutusunda görüntülenen soruyu veya istemi temsil eder. İkinci parametre ise onay butonuna tıklandığında yapılacak işlemi belirtir. Reddet butonuna tıklanırsa fonksiyon üçüncü parametrede belirtilen eyleme geçer.

Genel kullanımı aşağıdaki gibidir.

```
Clomosy.AskAndCall('Çıkış yapmak istediğinize emin misiniz?','ShowMessage("Evete tıklandı")','ShowMessage("Hayıra tıklandı")');
```

2.6.4. InputAndCall

Kullanıcın veri girmesi için metin kutusu içermektedir. Veri girişi, mesaj kutusunu görüntüleyerek gerçekleştirilmektedir. Kullanıcıdan iletişim kutusuna giriş yapmasını istemenin yolunu sağlamaktadır.

InputAndCall dört parametreden oluşur. İlk parametre başlık değeridir. İkinci parametre, giriş alanında görüntülenecek varsayılan metni belirtmek için kullanılır. Üçüncü parametre, onay butonuna basıldığında tetiklenecek eylemi tanımlamalı, dördüncü parametre, iptal butonuna basıldığında tetiklenecek eylemi belirtmelidir.

```
var
 gelenMtn : String;

void girisSorgusunuGetir;
{
   gelenMtn = Clomosy.GlobalResult;
   ShowMessage(gelenMtn);
}

{
  InputAndCall('Adınızı giriniz','--- ','girisSorgusunuGetir','Exit');
}
```

Örnek Uygulamalar

1. Uygulama

İki tam sayı değişkeni tanımlayın ve büyüklüklerini karşılaştırın. Uygulama amacı, ilk sayı ikinci sayıdan büyükse **True**, aksi takdirde **False** değerini döndürmelidir.

2. Uygulama

İki string değişkeni tanımlayıp, bunları birleştirerek ekranda gösterin.

3. Uygulama

Bir öğrencinin vize ve final notlarını alarak ortalama hesaplayın ve bu sonucu ekranda gösteren bir program tasarlayın.

4.Uygulama:

Bir metin kutusundaki metni düzenleyerek, ikinci ve üçüncü kelimelerin yerlerini değiştirin.

İşlem: Düğmeye basıldığında, metin kutusundaki metnin ikinci ve üçüncü kelimelerini değiştiren ve metin kutusunun içeriğini düzenleyen bir program hazırlayın. Metin, tam olarak bir boşlukla ayrılmış üç kelime içermelidir.

5.Uygulama

Bir metin kutusundaki noktalama işaretlerinin sayısını sayın ve sonucu döndüren bir fonksiyon yazın.

6.Uygulama

Bir metin kutusuna girilen metindeki kelime sayısını hesaplayın.

7.Uygulama

Metin kutusuna girilen metni tersine çeviren ve sonucu gösteren bir program yazın.

BÖLÜM 3. Program Denetim Deyimleri

3.1. Karşılaştırma Deyimleri

3.1.1. if, if-else

Programda Koşullu Yürütme. IF …ELSE Deyimi

Bu bölüme kadar sıralı bir eylem süreci olan farklı programlar yazıldı. Yazılan programlarda, eylemler yazıldığı gibi birbiri ardına gerçekleştirildi. Bu program yapısına 'Lineer Program Yapısı' denilmektedir.

Gerçek hayatta, genellikle çeşitli seçeneklerle karşılaşılır. Dışarısı soğuksa kaban giyilir, değilse ceket giyilir. Bu durum bir akış şeması olarak sunulabilir:

Resim 37

Baklava şeklindeki kutunun içinde "dışarısı soğuk" ifadesinin doğruluğuna göre "ceket giy" ya da "kaban giy" eylemleri gerçekleştirilir.

Bu yapıya 'dallanma yapısı' denir.

Clomosy'de, 'If…Else' deyimi aşağıdaki gibi kullanılarak gerçekleştirilir.

If < mantıksal ifade >

<deyim 1>

Else

<deyim 2>;

If …Else'nin yürütme sırası:

İlk olarak, mantıksal ifade değerlendirilir,

Mantıksal ifade **True** olarak değerlendirilirse, deyim 1 yürütülür, değil ise (mantıksal ifade **False** olarak değerlendirilir), ifade 2 yürütülür.

Örnek 1.

İki tam sayı verildiğinde, bunların maksimum değerini hesaplayınız.

```
var
  a, b, m: integer;
{
  a = 20;
  b = 17;
  if (a>b)
    m = a
  else
    m = b;
  ShowMessage(' Maksimum değer: '+IntToStr(m));
}
```

If…Else deyiminin *Else* anahtar sözcüğünden önce noktalı virgüle ihtiyaç duymadığına dikkat edilmelidir, çünkü bu tek bir karmaşık ifadedir ve *If* olmadan *Else* olamaz.

Birkaç eylem yürütülmek istendiğinde, birleşik ifadeler kullanılmaktadır. Birleşik ifadenin işleç parantezi "{ ....}" içerisine alınmış birkaç ifadeden oluşmaktadır. Örneğin, koşullarda birden fazla ifadenin çalıştırılması gerekiyorsa, bu ifadeler işleç parantezleri içinde ({…}) yazılır:

```
…
If  (x<0)
{
        y = 1;
        k = k+1;
} else{
        y = 2*x;
        k = x;
}
```

If..Else If deyimi

If-else if ifadeleri, programlama dilinde koşullu dallanmayı sağlayan yapıları ifade eder. Bu yapılar, programın belirli koşullara göre farklı yollar izlemesini sağlar. Clomosy'de "if", "else if" ve "else" ifadeleriyle bu yapılar oluşturulur.

Genel bir *if-else if-else* kullanımı şu şekildedir:

```
if (koşul1)
{
    // koşul1 doğruysa yapılacak işlemler
}
else if (koşul2)
{
    // koşul1 yanlış, koşul2 doğruysa yapılacak işlemler
}
else if (koşul3)
{
    // koşul1 ve koşul2 yanlış, koşul3 doğruysa yapılacak işlemler
}
else
{
    // Hiçbir koşul doğru değilse yapılacak işlemler
};
```

Aşağıdaki örnekte değişkene atanan sayının kontrolü yapılmakta ve kısaca if-else if-else deyimi gösterilmektedir.

```
var
  sayi: Integer;
{
  sayi = 12;

  if (sayi > 0)
    ShowMessage('Girilen sayı pozitif.')
  else if (sayi < 0)
    ShowMessage('Girilen sayı negatif.')
  else
    ShowMessage('Girilen sayı sıfır.');
}
```

İç içe If …Else deyimi

Else'den sonra gelen ifadeler de koşullu ifadeler olabilir. Bu durumda If …Else deyimine iç içe deyim denir.

```
If  <mantıksal ifade 1>

        <deyim 1>

Else

        If < mantıksal ifade 2>

                < deyim 2>

        Else

                < deyim 3>;
```

Birden fazla talimat yerleştirmek için ise:

```
If  < mantıksal ifade 1>
{
        < ifade 1>
        If < mantıksal ifade 2>
        {
                < ifade 2>
                < ifade 3>
        }
        Else
                < ifade 4>;
}
Else
{
        If < mantıksal ifade 3>
                < ifade 5>
        Else
        {
                < ifade 6>;
                 < ifade 7>;
        }
}
```

Else neye ait olduğu (ilk Else veya ikinci Else) net olmadığı için belirsiz görünebilir. Else'nin en yakın If'e ait olduğu kuralı vardır.

Karışıklığı ve olası hataları önlemek için, iç içe If … Else bloklar ({ … }) içerisine koyulması tercih edilir.

3.1.2. case

Bir değişkenin içeriğine bakarak, programın akışını birçok seçenekten birine yönlendiren bir karşılaştırma deyimidir. Deyimin genel yazımı şu şekildedir.

```
Case (degisken) of
{
        Sabit1 :
                {
                .. Küme1;  ..
                }
        ….
        SabitN :
                {
                .. KümeN;  ..
                }
        else
                {
                .. Küme_Default;  ..
                }
}
```

Değişkenin tuttuğu değer hangi sabit ile eşleşiyorsa, ilgili küme ve arkasındaki tüm kümeler yürütülür. Ancak, küme deyimleri arasında bir break bulunuyorsa, sonraki tüm deyimler ve fonksiyonlar atlanarak, 'Case' bloğunun sonuna gidilir.

Yukarıda verilen örnekte değişkene atanmış sabit değerle şart olarak koyulan sabit değerler sırayla karşılaştırılmaktadır. Eğer eşitlik sağlanırsa, o sabite ait deyimler ve sonra gelen deyimler yürütülür; sağlanmaz ise else anahtar sözcüğündeki deyimler yürütülür. En son kapatma küme karakteri (}) case deyiminin sonunu gösterir. Sondaki *else* anahtar sözcüğünün kullanımı isteğe bağlıdır; kullanılmazsa ve değişkenin içeriği hiçbir sabit değere eşit değilse, *case* içindeki hiçbir deyim yürütülmez.

Aşağıda örnekte; ilk önce bir değişken (sayi) oluşturuldu ve bu değişkene değer ataması yapıldı. Ardından *case* deyiminde bu değişkenin değeri karşılaştırıldı. Değer 0 ve 1 değerine sahip değil ise default kısmı olan else bloğuna yönlendirilir ve işlem sona erer.

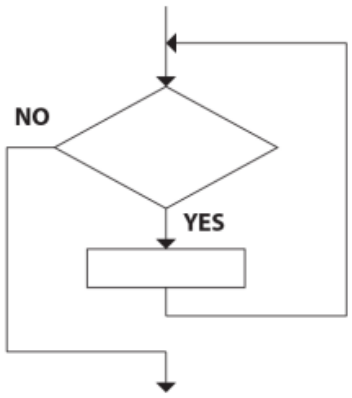
```
var
 sayi:Integer;
{
  sayi=3;

  case sayi of
  {
  0 : ShowMessage('True');
  1 : ShowMessage('False');
  else
  ShowMessage('Bu değer geçersizdir.');
  }
}
```

3.2. Döngü Deyimleri

Döngü, tekrar tekrar yürütülen bir ifade dizisidir. Clomosy' de üç döngü türü vardır: Sınırlı sayıda yürütülen döngü (for döngü), ön koşullu döngü (while döngü) ve son koşul döngüsü (repeat-Until).

3.2.1. While Döngüsü

While döngüsü, koşullu ifade True ise aynı ifadeyi tekrar tekrar yürütür. Bir koşullu ifade genellikle döngü içinde değişir ve deyim yürütülmeden önce hesaplanır. Bu nedenle, ifade değeri en baştan False ise, ifade hiç yürütülmez.



Resim 38

Birden fazla deyim yürütmeniz gerekiyorsa, bloklarının açılması gerekir (**{…}**). Deyimin genel yazımı şu şekildedir.

```
while <mantıksal ifade>
{
        <ifade 1>;
        <ifade 2>;
        <ifade 3>;
}
```

Bir örnek ile faiz hesaplama uygulaması yapalım.

Amaç:

Başlangıçta 1000 TL ile, yıllık %75 faiz oranı uygulayarak, paranın değerinin 50.000 TL'ye ulaştığı ya da geçtiği ilk yılı hesaplayın.

Koşullar:

Başlangıç miktarı: 1000 TL.

Yıllık faiz oranı: %75.

Hedef miktar: 50.000 TL.

İşlem:

'Para' adlı değişkeni başlangıç miktarı olan 1000 TL olarak tanımlayın. Her yıl için, 'Para' değişkenini %75 oranında artırın. 'Para' değişkeninin değeri 50.000 TL'ye eşit veya bu değeri aştığında, döngüyü sonlandırın. Döngü sonlandığında, 1000 TL'nin kaç yıl sonra 50.000 TL'ye ulaştığını ve son değerini ekranda gösterin.

```
Var
 Yil:Integer;
 Para : Float;
{
  Yil = 0;
  Para=1000;
   while (Para < 50000)
  {
```

```
    Para = Para + (Para*0.75);
    Yil = Yil +1;
  }
  ShowMessage('1000 TL. '+IntToStr(Yil)+' yıl sonra '+IntToStr(Para)+' TL olur.');
}
```

3.2.2. For Döngüsü

Diğer döngü deyimleri gibi bir öbeği birçok kez tekrarlamakta kullanılır. Koşul sınanması *while*'da olduğu gibi çevrime girmeden yapılır. Diğer döngülerden farkları vardır: bunlar, döngü sayacı olması ve ona başlangıç değeri verilmesi ve bir diğer fark ise döngü sayacının arttırılmasıdır. Genel yazım biçimi:

```
For ( <döngü sayacı> = <başlangıç değeri> to <son değer> )

{

        < ifade >;

}
```

Ya da

```
For ( <döngü sayacı> = <başlangıç değeri> downto <son değer> )

{

          < ifade >;

}
```

<döngü sayacı> Tam sayı türünde bir değişken, *to* durumunda +1 ve *downto* durumunda -1 adımıyla baştan sona bir değer kabul eder. Döngü sayacının değeri otomatik olarak değişir.

<başlangıç değeri> Herhangi bir tam sayı ifadesi; döngü sayacının başlangıç değeridir.

< son değer > Herhangi bir tam sayı ifadesi; döngü sayacının son değeridir.

<ifade> İfade, döngü sayacı son değerden büyük (*to* durumunda) veya küçük (*downto* durumunda) kadar yürütülür.

Başlangıç ve son değerler, döngü girişinde yalnızca bir kez hesaplanır.

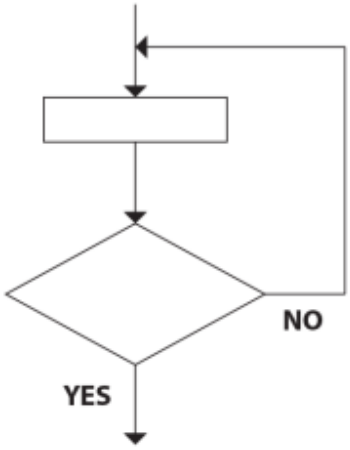
Örnek olarak; 1-10 arası sayıları, karesi ile yazdıralım.

```
var
 sayi: Integer;
{
  for (sayi = 1 to 10)
  {
   ShowMessage(IntTostr(sayi)+' sayısının karesi = '+ IntTostr(sayi*sayi));
  }}
```

3.2.3. Repeat - Until Döngüsü

Bu deyimin *while*'dan farkı, koşulun döngü sonunda sınanmasıdır. Yani koşul sınanmadan çevrime girilir ve döngü kümesi en az bir kez yürütülür. Bu döngü yapısı *Repeat-Until* olarak yazılmaktadır.

Koşul sağlandığında döngü sonlanır. Yani, koşul True olduğunda döngüden çıkılır.



Resim 39

Genel yazım biçimi:

```
repeat {
        < ifade 1>;
        …
} until ( <mantıksal ifade> );
```

Örnekte, döngü içinde rastgele bir sayı üretilir. Eğer bu sayı belirlenen koşulu karşılamıyorsa, döngü tekrar bir rastgele sayı üretir ve bu süreç, koşul sağlanana kadar devam eder.

```
var
  sayi:Float;
{
  repeat
    sayi = Random() * 70;
      ShowMessage('Rastgele alınan değer istenilen aralıktadır : '+ IntToStr(sayi));
    until(sayi > 50);
      ShowMessage('Döngüden çıkıldı. Sayı :'+ IntToStr(sayi));
}
```

3.3. Döngü Yönlendirme İfadeleri

Döngü ifadelerinde, çevrimi acil sonlandırması gerektiğinde veya döngünün bir sonraki çevrime geçmesi istendiğinde break ve continue kullanılır.

3.3.1. Break

Bir döngü içinde "break" ifadesiyle karşılaşıldığında, döngü hemen sonlanır ve program akışı, döngüden sonraki ilk ifadeye ya da fonksiyona atlar. Bu ifade, özel bir durum ortaya çıktığında while, for, repeat-until döngüsünden çıkmak için kullanılır. İç içe döngülerde kullanıldığında, en içteki döngüden çıkılır.

Aşağıdaki örnek incelendiğinde;

```
var
  i: Integer;

{
  for (i = 1 to 10)
  {
```

```
    ShowMessage('Değer : '+ IntToStr(i));
    // i değeri 5 olduğunda döngüden çık
    if (i == 5)
    {
      ShowMessage('i değeri 5 olduğu için döngüden çıkıyorum.');
      Break;
    }
  }
  ShowMessage('Döngü sonlandı.');
}
```

For döngüsü, koşulun 1'den 10'a kadar dönmesini sağlarken, 5 değerine ulaşıldığında "break" komutu kullanılarak 'for' döngüsü aniden sonlandırılır. Bu durumda döngü, bir sonraki değere geçmeden sona erer.

3.3.2. Continue

Continue ifadesi, bir döngü içinde bulunduğu anın geri kalanını atlamak ve döngüyü bir sonraki iterasyona geçirmek için kullanılır. *break* deyiminin tersidir. *break* döngüyü ivedi olarak sonlandırır, *continue* bir sonraki çevrime geçirir. Genelde kullanım biçimi,

```
.
for (başlangıç to koşul) {
.
if (…) continue;   //koşul olumlu ise aşağısı atlanır
.
}
```

olarak verilebilir. Bir döngü ifadesi içinde 'continue' ile karşılaşıldığında sonraki ifade ve fonksiyonlar atlanarak bir sonraki çevrim için koşul sınaması yapılır. Örneğin,

```
var
  i: Integer;
{
  for (i = 1 to 10)
  {
    // i değeri 5 olduğunda bu iterasyonu atla, diğer iterasyona geç
    if (i == 5)
    {
      ShowMessage('i değeri 5 olduğu için bu iterasyonu atlıyorum.');
      Continue;
    }
    // 5 değeri haricindeki diğer i değerlerini ekrana yazdırır
    ShowMessage('Döngüdeyim, i = '+ IntToStr(i));
  }
}
```

Programında bir döngü vardır ve bu döngü 1'den 10'a kadar dönmektedir. Programda i değişkeni 5 değerine geldiğinde *if* şartından dolayı bu değeri atlar ve ardından gelen kod parçalarını da işleme almadan döngünün bir sonraki değerine geçer.

3.3.3. Exit

Bulunduğu satırdan itibaren, lokal metot içerisinden çıkmayı sağlar. Döngü kontrol ifadesi olmamakla birlikte, döngü içerisinde kullanılması durumunda hem döngüyü sonlandıracak hem de bulunduğu lokal metot içerisinden çıkacaktır. Break ifadesinden farklı olarak, döngüden sonraki lokal metot kod satırları da çalıştırılamaz.

```
var
  i: Integer;
{
  for (i = 1 to 10)
  {
```

```
    ShowMessage('Değer : '+ IntToStr(i));
    if (i == 5)
    {
      ShowMessage('i değeri 5 olduğu için döngüden çıkıyorum.');
      Exit;
    }
  }
  ShowMessage('Döngü sonlandı.');
}
```

Örnek Uygulamalar

1. Uygulama

Kullanıcıdan alınan iki tam sayı 'm' ve 'n' için, eğer 'n', 'm'ye tam olarak bölünebiliyorsa, bölme işleminin sonucunu gösterin. Bölünemiyorsa, “n, m'ye bölünemez” mesajını gösterin.

2. Uygulama

Kullanıcıdan alınan üç basamaklı bir tam sayının palindrom olup olmadığını kontrol edin. Bir sayı palindrom ise, hem sağdan sola hem de soldan sağa aynı şekilde okunur.

3. Uygulama

Kullanıcıdan alınan üç farklı gerçek sayı arasından maksimum ve minimum değerleri bulun.

4. Uygulama

Kullanıcı tarafından girilen A, B ve C katsayılarına sahip Ax^2 + Bx + C = 0 ikinci dereceden denkleminin köklerini bulun. Eğer denklemin geçerli kökleri yoksa, ilgili mesajı görüntüleyin. A≠0 olmalıdır.

5. Uygulama

Kullanıcıdan alınan bir yılın artık yıl olup olmadığını kontrol edin. Bir yıl, eğer 4'e bölünebiliyorsa genellikle artık yıldır; ancak 100'e bölünebilen yıllar arasında, sadece 400'e bölünebilenler artık yıldır.

Örneğin; 1700, 1800 ve 1900 artık yıl değil, 2000 ise artık yıl.

BÖLÜM 4. Prosedürler

Programları etkili bir şekilde yazmak, sadece programlama dillerinin operatörlerini bilmekle sınırlı değildir. Yapılandırılmış, prosedürel programlama becerileri ve ilkeleri konusuna da hâkim olmak gereklidir.

Bu ilkelerin pratik uygulaması:

- Programları kolayca yazmanızı,
- Anlaşılır ve okunabilir programlar oluşturmanızı,
- Program hatalarını daha hızlı bulmanızı,
- Programları daha hızlı değiştirip iyileştirmenizi sağlar.

Yapılandırılmış programlamanın ana ilkelerinden biri yukarıdan aşağıya programlama olarak adlandırılır. Bu ilke, herhangi bir programın ana görevinin birkaç alt göreve bölünmesi gerektiğini belirtir. Ardından, her alt görev daha da basit alt görevlere bölünmelidir ve bu süreç, uygulaması kolay bir dizi küçük, basit görev elde edene kadar devam ettirilmelidir.

Ayrı alt görevler, dil yapısında void olarak tanımlanır.

Prosedür *(void),* parametreli veya parametresiz olarak kullanılabilir. Parametreli prosedür *(void),* bir veya daha fazla parametre alabilir. Bu parametreler prosedürün içinde kullanılabilir ve bir değer döndürmemektedir. Parametresiz prosedürler ise hiçbir parametre almaz.

Parametresiz Prosedür (Void) Tanımı:

```
void <void ismi>;
  < tanımlama >
{
  <ifadeler>
}
```

Parametreli Prosedür (Void) Tanımı:

```
void <void ismi> (< arama parametreleri listesi >);
```

Prosedürler, kodunuzda kullanılmadan önce tanımlandıkları yere, yani çağrıldıkları kod bloğundan önce uygulama bölümünde tanımlanmalıdır. Bu, prosedürün kullanılacağı yerlerde önceden bilinmesini ve anlaşılmasını sağlar.

```
void  < void  ismi> (<parametre listesi>);

          < tanımlama >

{

          <ifadeler>

}
```

Örneğin, girilen bir sayının faktöriyelini hesaplayan ve sonucu ekrana yazdıran bir parametreli prosedür tanımlayalım.

Prosedür Tanımı; Parametre olarak bir tamsayı alan ve bu sayının faktöriyelini hesaplayarak ekrana yazdıran bir prosedür tanımlanacaktır.

```
// Parametreli bir prosedür tanımı
void FaktoriyelHesapla(sayi: Integer);
var
  i, faktoriyel: Integer;
{
  faktoriyel = 1;

  // Faktöriyel hesaplama
  for (i = 1 to sayi)
  faktoriyel = faktoriyel * i;

  // Sonucu ekrana yazdır
  ShowMessage(IntToStr(sayi) + ' sayısının faktöriyeli: '+ IntToStr(faktoriyel));
}

var
  girilenSayi: Integer;
{
  girilenSayi = 5;

  // Faktöriyel hesaplama prosedürünü çağır
  FaktoriyelHesapla(girilenSayi);
}
```

Açıklama:

Bu programda, bir sayı girilir ve bu sayının faktöriyeli hesaplanır. Parametreli *FaktoriyelHesapla* prosedürü, aldığı sayının faktöriyelini hesaplar ve sonucu ekrana yazdırır. Ardından, kullanıcının girdiği sayıyı bu prosedürle işleme sokarak faktöriyeli hesaplanır. Bu şekilde, parametreli prosedürün kullanımı kodun daha modüler olmasını sağlar ve kod tekrarını azaltır.

Örnek Uygulamalar

1. Uygulama

A ve B değişkenleri arasında değer alışverişi yapın ve C ile D değişkenlerinin değerlerini değiştiren bir program oluşturun.

Yapılacaklar:

İki değişkenin değerlerini değiştirecek bir prosedür tanımlayın.

Bu prosedür, A ve B değişkenlerinin değerlerini birbiriyle değiştirmeli ve aynı şekilde C ve D için de bu işlemi yapmalıdır.

2. Uygulama

İki üçgenin kenar uzunlukları verildiğinde, bu üçgenlerin çevrelerinin ve alanlarının toplamını bulun.

Yapılacaklar:

Üçgenin çevresini ve alanını hesaplayacak bir prosedür tanımlayın.

Bu prosedür, verilen kenar uzunluklarına göre her üçgen için çevre ve alan hesaplaması yapmalı ve bu değerlerin toplamını döndürmelidir.

3. Uygulama

İç yarıçapı R1 ve dış yarıçapı R2 olan bir halkanın alanını hesaplayın.

Yapılacaklar:

Bir dairenin alanını hesaplamak için bir prosedür tanımlayın.

Bu prosedür, verilen yarıçap değerleri ile dairenin alanını hesaplamalıdır.

Halkanın alanını hesaplamak için, büyük dairenin alanından küçük dairenin alanını çıkarın.

BÖLÜM 5. Fonksiyonlar

Clomosy dilinde bir fonksiyon, belirli bir görevi yerine getirmek için tasarlanmış bir alt program parçasıdır. Fonksiyonlar, belirli bir giriş (parametreler) alır, bu girişlere dayanarak bir işlem gerçekleştirir ve bir çıktı (Result) döndürür. Fonksiyonlar, tekrar kullanılabilirlik ve kodun modüler hale getirilmesi için önemlidir.

Fonksiyonlar, yürütme ifadelerine ek olarak bir değer döndürmeleri dışında prosedürlerle (void) aynıdır.

Fonksiyon bildirim biçimi:

```
Function <fonksiyon ismi> (<parametre listesi >): <dönüş değeri türü >;
        < yerel bildirimler >
{
        < ifadeler >
}
```

Fonksiyon hem değer parametrelerine hem de değişken parametrelere sahiptir. Parametreler, prosedürün parametreleri için geçerli olan aynı kurallara tabiidir.

Fonksiyon, işlev adına bir değer atayan en az bir atama işlevine sahip olmalıdır.

Örnek;

Trigonometrik bir fonksiyon kullanarak dik üçgenlerde bir kenarın uzunluğunu hesaplamak için program yazalım. Bunun için **Tg(A)** fonksiyonu oluşturulur. Bu fonksiyon alfa değerini parametre olarak alır ve tan(alfa) değerini döndürür.

Bu kod, trigonometrik fonksiyonların nasıl kullanılacağını gösterir ve dik üçgenlerde kenar uzunluklarını hesaplama işlemini gerçekleştirir.

```
var
  a, b, alfa: real;

Function tgHesapla(x: real): real;
var
  y: real;
Const
  Pi = 3.14;
```

```
{
  y= x / 180 * pi;
  Result = sin(y) / cos(y);
}


{
  alfa = 25.50;
  a = 20;
  b = a * tgHesapla(alfa);
  ShowMessage('Sonuç: '+FloatToStr(b));
}
```

BÖLÜM 6. Hazır Kütüphane Fonksiyonları

Clomosy mobil geliştirme ortamında program yazılmasını kolaylaştırmak ve sıkça kullanılan program parçalarını hazır bulundurmak amacıyla kütüphane fonksiyonlarına sahiptir. Kütüphane fonksiyonları, konularına göre sınıflanır ve anlamlı bir adlandırma ile tanımlanır. Örneğin; 'cos, abs, sqrt' gibi matematiksel işlemlerle ilgili fonksiyonlar bulunurken, tarih ve saat fonksiyonları gibi gerekli fonksiyonlar bulunur. Bu bölümde Clomosy'nin desteklediği kütüphane fonksiyonlarına değinilecektir.

Clomosy'de fonksiyonları kullanabilmek için program içerisine bir kütüphane eklemesi yapılmasına gerek yoktur. Sadece fonksiyonun yazılması ve kullanılması yeterlidir.

6.1. Karakterler Üzerine İşlem Yapan Fonksiyonlar

Tek bir karakter üzerine işlem yapan bu makro fonksiyonlar sorgulama ve değiştirme olarak ikiye ayrılabilir. Sorgulama fonksiyonları, bir karakterin kullanılan karakter kümesindeki sınıfını belirler; olumlu olduğu zaman 0 veya 0'dan büyük, olumsuz ise 0'dan küçük bir sayı döndürülmektedir.

LowerCase()

Tüm harflerin küçük harfe dönüştürüldüğü bir String kopyası oluşturur.

```
var
   basitStr : String;
{
   basitStr = LowerCase('CLOMOSY'i öğreniyorum.');
   ShowMessage(basitStr);
}
```

UpperCase()

Tüm harflerin büyük harfe dönüştürüldüğü bir String kopyası oluşturur.

```
var
 basitStr : String;


{
 basitStr = UpperCase('CLOMOSY'i öğreniyorum.');
 ShowMessage(basitStr);
}
```

Copy()

Kopyalama işlevi mevcut bir dizinin bir kısmından yeni bir dizi oluşturur. Bu fonksiyon üç parametre almaktadır. İlk parametreye kopyalanacak metin girilir, ikinci parametre başlangıç metindeki başlangıç indisini belirler, burada metnin ilk karakteri 1 olarak başlamaktadır. Son parametre ise kaç karakter alınıp kopyalanacağı anlamına gelmektedir.

Örnekte, metnimizin içerisindeki string değer '12345678' olmaktadır. Burada '3456' metni kopyalanmak istenmektedir. Bunun için fonksiyon içerisindeki ikinci parametre 3.karakterden başlamalı ve 4 karakter alınmalıdır. Bunun için aşağıdaki gibi bir tanımlama yapılmalıdır.

```
var
  kaynakMetin, hedefMetin : string;
{
  kaynakMetin = '12345678';
  hedefMetin = Copy(kaynakMetin, 3, 4);
  ShowMessage('Hedef metin: ' + hedefMetin);
}
```

Trim()

Bir dizenin başındaki ve sonundaki boş ve kontrol karakterlerini (satır besleme gibi) kaldırır.

TrimLeft()

Bir dizenin başındaki boş ve kontrol karakterlerini (satır besleme gibi) kaldırır.

TrimRight()

Bir dizenin sonundaki boş ve kontrol karakterlerini (satır besleme gibi) kaldırır.

Kullanımı:

```
const
  textValue = '      Letters     ';
{
  ShowMessage('/'+TrimLeft(textValue)+'/');
  ShowMessage('/'+TrimRight(textValue)+'/');
  ShowMessage('/'+Trim(textValue)+'/');
}
```

Çıktısı şu şekilde olmaktadır:

```
/Letters     /
/     Letters/
/Letters/
```

Delete()

Delete fonksiyonu, mevcut bir dizinin bir kısmını silmektedir. Bu fonksiyon üç parametre almaktadır. İlk parametreye silinecek metin girilir, ikinci parametre başlangıç metindeki başlangıç indisini belirler. Burada metnin ilk karakteri 1 olarak başlamaktadır. Son parametre ise kaç karakter alınıp silineceği anlamına gelmektedir.

Örnekte, metnimizin içerisindeki string değer '12345678' olmaktadır. Burada '3456' metni silinmek istenmektedir. Bunun için fonksiyon içerisindeki ikinci parametre 3.karakterden başlamalı ve 4 karakter alınmalıdır. Aşağıdaki gibi bir tanımlama yapılmalıdır.

```
var
  kaynakMetin : string;
{
  kaynakMetin = '12345678';
  Delete(kaynakMetin, 3, 4);
  ShowMessage('Hedef metin: ' + kaynakMetin);
}
```

Sonuç olarak silinen değerlerden sonra kalan yeni metin '1278' olur.

Insert()

Insert fonksiyonu, bir karakter dizesinin belirli bir konumuna başka bir karakter dizesi ekler. Bu işlev, hedef karakter dizesinin belirli bir pozisyonuna bir veya daha fazla karakter eklemek için kullanılır.

Insert fonksiyonunun imzası şu şekildedir:

```
function Insert(const SubStr: string; const TargetStr: string; Index: Integer);
```

Bu fonksiyon üç parametre almaktadır. İlk parametreye eklemek istenilen string ifade edilir, ikinci parametre hedef stringi son parametre ise hedef stringin hangi konumuna ekleneceğini belli eden index numarası yazılır. Burada metnin ilk karakteri 1 olarak başlamaktadır. Son parametre ise kaç karakter alınıp silineceği anlamına gelmektedir.

Örnekte, ana metin içerisindeki string değer 'Merhaba, dünya!' olmaktadır. Burada 'güzel ' alt metni 'Merhaba,*güzel* dünya!' olarak eklenmek istenmektedir. Bunun için fonksiyon içerisindeki üçüncü parametre 9.karakterden başlamalıdır.

```
var
  orijinalStr, altStr: string;
{
  orijinalStr = 'Merhaba, dünya!';
  altStr = 'güzel ';
  Insert(altStr,orijinalStr, 9);
  ShowMessage(orijinalStr); // "Merhaba,güzel dünya!"
}
```

Length()

Length fonksiyonu, bir stringin uzunluğunu döndüren bir fonksiyondur. Bir parametre almaktadır ve bu uzunluğu bulunmak istenen string verisidir.

```
function Length(const SourceString string):Integer;
```

Örneğin, Clomosy metninin uzunluğunun döndürülmesi isteniyorsa;

```
var
   metin    : string;
   uzunluk : Integer;

 {
   metin = 'Clomosy';
   uzunluk = Length(metin);
   ShowMessage(metin+' kelimesinin uzunluğu = '+IntToStr(uzunluk));
 }
```

AnsiCompareStr()

İki ANSI karakter dizgisini (string) karşılaştıran bir işlevdir. Karşılaştırma büyük/küçük harfe duyarlıdır.

```
var

  baslangicDeger,ikinciDeger : String;

  sonuc :Integer;

{

  baslangicDeger = 'Clomosy';

  ikinciDeger = 'Clomosy';

  sonuc = AnsiCompareStr(baslangicDeger, ikinciDeger);
```

```
  if (sonuc < 0)  ShowMessage(baslangicDeger +' < '+ ikinciDeger);
  if (sonuc == 0)  ShowMessage(baslangicDeger +' = '+ ikinciDeger);
  if (sonuc > 0)  ShowMessage(baslangicDeger +' > '+ ikinciDeger);
}
```

Yukarıdaki örnekte iki string değişken verilmiştir ve iki içerikte aynıdır. Burada sadece eşittire ait mesaj kutusu ekrana gelecektir.

AnsiCompareText()

İki ANSI karakter dizgisini (string) karşılaştıran bir işlevdir. Bu işlev, büyük-küçük harf duyarlılığı olmadan iki karakter dizgisini sıralar. Yani, iki karakter dizgisi arasındaki alfabetik sıralamayı belirlerken büyük ve küçük harf farkını dikkate almaz.

```
var
  baslangicDeger,ikinciDeger : String;
  sonuc :Integer;
{
  baslangicDeger = 'ClomoSY';
  ikinciDeger = 'Clomosy';
  sonuc = AnsiCompareText(baslangicDeger, ikinciDeger);

  if (sonuc < 0)  ShowMessage(baslangicDeger +' < '+ ikinciDeger);
  if (sonuc == 0)  ShowMessage(baslangicDeger +' = '+ ikinciDeger);
  if (sonuc > 0)  ShowMessage(baslangicDeger +' > '+ ikinciDeger);
}
```

Örnekte, harflerin büyük veya küçük olmasının bir önemi olmaksızın, sonuç olarak aynı şarta bağlı olarak bir mesaj kutusu görüntülenecektir.

6.2. Standart Matematik Fonksiyonları

Clomosy' in güçlü matematik kütüphanesi, geliştiricilere geniş bir matematiksel işlevsellik sağlar. Temel matematik operasyonlarından, trigonometrik fonksiyonlara, karekök

hesaplamalarından, üssel işlemlere kadar birçok özellik içerir. Bu kütüphane, matematiksel hesaplamaların yanı sıra bilimsel uygulamalarda da geliştiricilere geniş olanaklar sunar. Clomosy' in matematik kütüphanesi, kullanıcı dostu olup karmaşık matematiksel problemleri etkili bir şekilde çözmek için tasarlanmıştır. Matematiksel fonksiyonları kullanmak için kütüphane yüklemesi yapılmasına gerek yoktur.

Abs()

Bir tamsayının mutlak değerini döndürür.

```
var
  i : Integer;


{
  i = -1235;
  //i'nin mutlak değeri alınıp i'ye geri atanmaktadır
  i = Abs(i);
  ShowMessage('Sonrası: '+IntToStr(i));
}
```

Örnekte çıktı olarak 1235 sayısı dönmektedir.

ArcTan()

Trigonometrik bir fonksiyondur. Bu fonksiyon, verilen bir oranın ters tanjantını (arctangent) hesaplar. Matematiksel ifadesi şu şekildedir:

$$\text{ArcTan}(x)=\tan^{-1}(x)$$

Bu, belirli bir oranın açısını (radyan cinsinden) verir. 'ArcTan' fonksiyonu, genellikle trigonometrik hesaplamalarda veya geometrik problemlerde kullanılır.

```
var
  deger : single;
const
  PI = 3.14;
{
```

```
  deger = ArcTan(PI/2);
  ShowMessage('PI/2 nin ArcTan değeri = '+FloatToStr(deger));
}
```

Cos()

Cos işlevi, radyan olarak verilen bir sayı değerinin kosinüs değerini veren matematiksel bir işlevdir.

```
var
  deger : single;
const
  PI = 3.14;
{
  deger = Cos(PI/3);  // = 180/3 = 60 derece
  ShowMessage('Cos(PI/3) = '+FloatToStr(deger));
}
```

Dec()

Dec prosedürü kendisine iletilen integer tipindeki değişkenin değerini bir azalmaktadır. Örneğin, 23 sayısına Dec özelliği uygulandığı anda değişkenin değeri bir azalır ve 22 olur.

```
var
  sayi    : Integer;
{
  sayi = 23;
  ShowMessage('Sayı : '+IntToStr(sayi));
  Dec(sayi);
  ShowMessage('Sayı - 1 : '+IntToStr(sayi));
}
```

Div()

İki sayının bölünmesi sonucu tam sayı kısmını döndüren bir işlevdir. Genellikle, bölme işleminden elde edilen tam sayı sonucuyla ilgilenildiğinde kullanılır. Div işlevi, bölme işleminin tam sayı kısmını döndürmek üzere tasarlanmıştır.

```
var
  sayi : Integer;


{
  sayi = 27 Div 2;


  ShowMessage('27 div 2 = '+IntToStr(sayi));
}
```

Örnekte 27 sayısı 2'ye bölünmektedir. Sonuç olarak 13 değeri dönmektedir.

Frac()

Bir ondalık sayının ondalık kısmını döndüren bir fonksiyondur. Örnekte sonuç olarak 0,75 değerini döndürmektedir.

```
var
   sayiDegeri : Integer;


{
   sayiDegeri = 12.75;
   ShowMessage(' Frac(12.75) = '+FloatToStr(Frac(sayiDegeri)));
}
```

Inc()

Inc prosedürü kendisine iletilen integer tipindeki değişkenin değerini bir arttırmaktadır. Örneğin, 23 sayısına Inc özelliği uygulandığı anda değişkenin değeri bir artar ve 24 olur.

```
var
  sayi    : Integer;
{
  sayi = 23;
  ShowMessage('Sayı : '+IntToStr(sayi));
  Inc(sayi);
  ShowMessage('Sayı - 1 : '+IntToStr(sayi));
}
```

Ln()

Bir sayının doğal logaritmasını hesaplamak için kullanılır. Doğal logaritma, matematikte "e" tabanındaki logaritmadır. Yani logaritma tabanı olarak "e" (Euler sayısı) kullanılır.

```
var
   ondalikliSayi ,sayi : float;
{
  sayi  = 2.14;
  ondalikliSayi = Ln(sayi);
  ShowMessage('Ln('+FloatToStr(sayi)+') = '+FloatToStr(ondalikliSayi));
}
```

Örnekte 2.14 değerine sahip değişken, sonuç olarak 0,76080582903376 değeri döndürmektedir.

Mod()

Bir sayının diğer bir sayıya bölümünden kalanı hesaplamak için kullanılır.

```
var
  bolmeSayisi,bolen,kalan : Integer;
{
  bolmeSayisi = 20;
  bolen = 12;
  kalan = bolmeSayisi Mod bolen;
  ShowMessage(IntToStr(bolmeSayisi)+' Mod '+IntToStr(bolen)+' : '+IntToStr(kalan));
}
```

Odd()

Verilen sayının tek olup olmadığını kontrol eden bir fonksiyondur. Bu fonksiyondan değişken değeri tek ise true değeri döndürür, çift ise false değeri döndürmektedir.

```
var
  i : Integer;
{
  i = 3;
  if (Odd(i)) ShowMessage('i değişkeni tek sayıdır');
}
```

Random()

Bu fonksiyon 0 ve 1 arasında rastgele bir değer üretmektedir.

```
x = Random();
```

Tam sayı değeri almak istiyorsanız aşağıdaki gibi tanımlayın. Bu işlem 0 ile 100 arasında rastgele bir değer döndürecektir.

```
x = Random() * 100;
```

Round()

Ondalıklı sayıları en yakın tam sayıya yuvarlamak için kullanılır. Bu fonksiyon, ondalık kısmı 0.5 ve daha yüksek olan sayıları bir üste, 0.5'ten küçük olanları bir alta yuvarlar.

```
var
  sayi : Float;
{
  sayi = 15.50;
  ShowMessage('Round('+FloatToStr(sayi)+') = '+FloatToStr(Round(sayi)));
}
```

Sqrt()

Pozitif bir değerin karekökünü bulmak için kullanılır. Sayı bir tam sayı veya kayan nokta tipi olabilir.

```
var
  I,sqrtI : Integer;
{
  I = 16;
  sqrtI = Sqrt(I);
  ShowMessage('Sqrt ('+IntToStr(I)+') : '+IntToStr(sqrtI));
}
```

Sin()

Sin fonksiyonu, radyan olarak verilen bir sayı değerinin sinüs değerini veren matematiksel bir işlevdir.

```
const
  PI = 3.14;
var
  sayiDegeri : float;
{
  sayiDegeri = Sin(PI/6);
  ShowMessage('Sin(PI/6) = '+FloatToStr(sayiDegeri));
}
```

Sqr()

Sqr fonksiyonu bir sayının karesini döndürür. Sayı bir tam sayı veya kayan nokta tipi olabilir.

```
var
   I,sqrI : float;
{
   I = 4.5;
   sqrI = Sqr(I);
   ShowMessage('Sqr ('+FloatToStr(I)+'):'+FloatToStr(sqrI));
}
```

Trunc()

Noktalı sayının tam sayı kısmını döndürür.

```
var
   valueNumber : Float;
{
  valueNumber = 12.60;
  ShowMessage('Trunc('+ FloatToStr(valueNumber) +') = '+IntToStr(Trunc(valueNumber))); }
```

Örnekte 12.60 değerine sahip değişken sonuç olarak 12 değerini döndürmektedir.

6.3. Zaman ve Tarih Fonksiyonları

Clomosy programlama dilinde, zaman ve tarihle ilgili işlemleri gerçekleştirmek için çeşitli fonksiyonlar bulunmaktadır. Bu fonksiyonlar, sistem saati, tarih aralıkları, saat, dakika, saniye gibi bileşenlere erişim ve manipülasyon sağlamaktadır. Bu sayede program geliştiricileri, zamanla ilgili işlemleri kolayca yönetebilmektedir.

Tarih ve zaman ile çalışmak için *TclDatetime* veri türünün kullanılması gerekmektedir.

```
Var
  a, b : TclDateTime;
```

Date

Mevcut sistem tarihini döndüren bir fonksiyondur. Bu fonksiyon, yalnızca gün, ay ve yıl bilgilerini içeren bir *TclDateTime* değeri döndürür, saat bilgisini içermez.

```
var
  gecerliTarih: TclDateTime;
{
  gecerliTarih = Date;
  // gecerliTarih, şu anki tarihi içerir (saat bilgisi olmadan)
  ShowMessage(gecerliTarih);
}
```

Date fonksiyonu, genellikle bir program içinde belirli bir anın tarihini almak veya karşılaştırmak için kullanılmaktadır. Sadece tarih bilgisine ihtiyacınız varsa ve saati dikkate almak istemiyorsanız, Date fonksiyonu kullanışlı bir araçtır. Örnek çıktı değeri ‘8.01.2024’ olabilir.

Time

Mevcut sistem saatinin tam saat, dakika, saniye bilgilerini içeren *TclDateTime* değeri döndüren bir fonksiyondur. Bu fonksiyon, yalnızca saat bilgisini içerir, tarih bilgisini içermez. Örnek çıktı değeri ‘16:35:49’ olabilir.

```
var
  gecerliSaat: TclDateTime;
{
  gecerliSaat = Time;
  // gecerliSaat, şu anki saati içerir (tarih bilgisi olmadan)
  ShowMessage(gecerliSaat)
}
```

Now

Mevcut sistem tarihini ve saatinin tam bilgisini içeren *TclDateTime* değeri döndüren bir fonksiyondur. Bu fonksiyon hem tarih hem de saat bilgisini içermektedir. Örnek çıktı değeri '8.01.2024 16:56:03' olabilir.

```
var
  gecerliTarihSaat: TclDateTime;
{
  gecerliTarihSaat = Now;
  ShowMessage(gecerliTarihSaat)
}
```

DayOfWeek()

DayOfWeek fonksiyonu, belirtilen bir tarihin, haftanın hangi gününe denk geldiğini belirlemek için kullanılmaktadır. Bu fonksiyon, bir *TclDateTime* değerini alır ve o tarihin haftanın hangi günü olduğunu 1'den 7'ye kadar olan bir sayı olarak döndürmektedir. Haftanın günleri, Pazar'dan Cumartesi'ye kadar 1'den 7'ye kadar olan sayılarla temsil edilmektedir.

```
var
   gecerliTarih: TclDateTime;
   deger: Integer;
{
   gecerliTarih = Now;
   deger = DayOfWeek(gecerliTarih);
```

```
  // deger şimdi, belirtilen tarihin haftanın hangi günü olduğunu içerir
  // 1: Pazar, 2: Pazartesi, ..., 7: Cumartesi
  ShowMessage(deger);
}
```

Bugünün pazartesi olduğunu düşünürsek sonuç olarak 2 değeri döndürür.

FormatDateTime()

FormatDateTime fonksiyonu, bir TclDateTime değerini belirli bir tarih/saat formatına göre biçimlendirmek için kullanılmaktadır. Bu fonksiyon, belirli bir biçim dizesi kullanarak tarihi veya saati istenen formatta bir metin haline getirmektedir.

Örnek kullanım:

```
var
  gecerliTarih: TclDateTime;
  yeniFormat: string;
{
  gecerliTarih = Now;
  // 'dd.mm.yyyy' formatında tarihi biçimlendirme
  yeniFormat = FormatDateTime('dd.mm.yyyy', gecerliTarih);
  ShowMessage('Biçimlendirilmiş Tarih: ' + yeniFormat);

  // 'hh:nn:ss' formatında saat ve dakikayı biçimlendirme
  yeniFormat = FormatDateTime('hh:nn:ss', gecerliTarih);
  ShowMessage('Biçimlendirilmiş Saat: ' + yeniFormat);
}
```

Bu örnekte, FormatDateTime fonksiyonuyla tarih 'dd.mm.yyyy' formatına ve saat 'hh:nn:ss' formatına göre biçimlendirildi. İlgili biçim dizesi içinde kullanılan 'dd', 'mm', 'yyyy', 'hh', 'nn', ve 'ss' gibi semboller, tarih ve saat bileşenlerini temsil etmektedir.

Bu fonksiyon, tarih/saat biçimleme işlemlerinde kullanıcıya esneklik sağlayarak ve istenilen formatta metin çıktısı almayı mümkün kılmaktadır. Ayrıntılı bilgilendirme almak için https://www.docs.clomosy.com sitesine bakabilirsiniz.

IncMonth()

Verilen bir TclDateTime değerine belirli bir sayıda ay eklemek için kullanılmaktadır. Bu fonksiyon, tarih ve saat bileşenlerini korurken ay bazında artış yapmaktadır. Artış değeri isteğe bağlıdır, varsayılan olarak 1'dir.

```
var
  tarihDegeri : TclDateTime;
{
  tarihDegeri = StrToDate('08 10 2023');
  ShowMessage('tarihDegeri = '+DateToStr(tarihDegeri));

  tarihDegeri = IncMonth(tarihDegeri, 3);
  ShowMessage('tarihDegeri + 3 ay = '+DateToStr(tarihDegeri));
}
```

Örnekte 3 ay eklendiğinde tarih ‘8.01.2024’ olarak gösterilmektedir.

EncodeDate

EncodeDate işlevi, geçirilen Yıl, Ay ve Gün değerlerinden bir TclDateTime dönüş değeri üretmektedir.

İzin verilen parametre değerleri şunlardır:

Yıl = 1..9999

Ay = 1..12

Gün = 1..31 (ay/yıla bağlı olarak)

```
var
   myDate : TclDateTime;
```

```
{
  myDate = EncodeDate(2023, 02, 27);
  ShowMessage('Date set to '+DateToStr(myDate));
}
```

EncodeTime

EncodeTime işlevi, geçirilen saat, dakika, saniye ve milisaniye değerlerinden bir TclDateTime dönüş değeri sağlamaktadır.

İzin verilen parametre değerleri şunlardır:

Saat = 0..23

Dakika = 0..59

Saniye = 0..59

Milisaniye = 0..999

DecodeDate

DecodeDate, bir TclDateTime değerini alır ve bu değeri yıl, ay ve gün olarak ayrı değişkenlere ayırmaktadır. Yani, TclDateTime türündeki bir tarihi parçalara ayırarak yıl, ay ve gün değerlerini elde etmenizi sağlamaktadır.

```
var
  myDate : TclDateTime;
  yil, ay, gun : Word;
{
  // Bir TclDateTime değeri oluşturma
  myDate = Now;

  // Tarihi parçalara ayırma
  DecodeDate(myDate, yil, ay, gun);
  // Elde edilen değerleri ekrana yazdırma
  ShowMessage('yil: ' + IntToStr(yil) + ', ay: ' + IntToStr(ay) + ', gun: ' + IntToStr(gun));
}
```

Bu örnekte, Now fonksiyonu ile güncel bir TclDateTime değeri alınır, ardından DecodeDate fonksiyonu kullanılarak bu değer yıl, ay ve gün olarak ayrılır. Ayrılan bu değerler daha sonra ekrana yazdırılır. Bu sayede bir TclDateTime değerinin içindeki yıl, ay ve gün bilgilerini ayrı ayrı elde etmiş olursunuz.

DecodeTime

DecodeTime, bir TclDateTime değerini alarak bu değeri saat, dakika, saniye ve milisaniye şeklinde ayrı değişkenlere ayırmaktadır. Yani, TclDateTime türündeki bir zamanı parçalara ayırarak saat, dakika, saniye ve milisaniye değerlerini elde etmenizi sağlamaktadır.

```
var
  saat, dakika, saniye, miliSaniye : Word;


{
  DecodeTime(Now, saat, dakika, saniye, miliSaniye);
  ShowMessage('Şimdi = '+TimeToStr(Now));
  ShowMessage('saat     = '+IntToStr(saat));
  ShowMessage('dakika   = '+IntToStr(dakika));
  ShowMessage('saniye   = '+IntToStr(saniye));
  ShowMessage('miliSaniye = '+IntToStr(miliSaniye));
}
```

Yukarıdaki örnekte, Now fonksiyonu ile güncel bir TclDateTime değeri alınır, ardından DecodeTime fonksiyonu kullanılarak bu değer saat, dakika, saniye ve milisaniye olarak ayrılır. Ayrılan bu değerler daha sonra ekrana yazdırılır.

Örnek Uygulamalar

1. Uygulama

Verilen bir gerçek sayının tam ve kesirli kısımlarını ayrı ayrı gösteriniz.

2. Uygulama

Girilen üç sayının toplamını ve çarpımını hesaplayarak görüntüleyiniz.

3. Uygulama

Üçgen, köşelerinin koordinatlarıyla belirtilir. Üçgenin çevresini ve alanını hesaplayınız ve görüntüleyiniz.

4. Uygulama

Dersleri eşit süreli olan bir okulda, derslerin süresinden farklı olarak eşit sürede mola veriliyor. İlk derse başlama saati, ders süresi, mola süresi ve ders sayısı metin kutularına girilerek, okul programı aşağıdaki gibi sunulmalıdır.

	Başlangıç Saati	Bitiş Saati
Ders 1	9:00	9:40
Teneffüs	9:40	9:50
Ders 2	9:50	10:30
Teneffüs	10:30	10:40
...		

Resim 40

5. Uygulama

Metin kutusuna bir tarih girilmeli. Bu tarihin, bugünden tam kaç yıl, ay ve gün ile ayrıldığını hesaplayınız. Sonucu farklı etiketlere çıktı olarak yazdırınız.

BÖLÜM 7. Diziler

Dizi, aynı veri türünden çok sayıda değişken anlamına gelmektedir. Dizi değişkenleri, sıra numarası olan bir liste gibi davranmaktadır. Dizi tek boyutlu olduğu gibi birden fazla boyuta da sahip olabilir. Her boyutun eleman sayısı ayrı ayrı belirlenmektedir. Dizi elemanlarına erişim, dizi adı ve indis değeri ile gerçekleştirilmektedir. İndis, bir elemanın dizi içerisinde sahip olduğu sıra numarasını ifade etmektedir. İlk elemanın indis numarası sıfırdır ve sonraki elemanların indis değeri de bir öncekine göre bir sayı artarak, son elemana kadar devam eder. İndis değeri köşeli parantezler arasına yazılmaktadır.

dizi1 [0], dizinin birinci elemanıdır.

dizi1 [1], dizinin ikinci elemanıdır.

dizi1 [2], dizinin üçüncü elemanıdır.

Diziler tanımı **array** ifadesinden sonra, boyut, uzunluk ve veri tipi belirtilerek tanımlanmaktadır.

```
var
  dizi1: array [0..10] of String;
```

Yukarıdaki örnek uygulamada tek boyutlu, 11 elemanlı ve string veri tipinde bir dizi tanımlanmıştır.

Aşağıdaki kod satırlarında bir dizi elemanına nasıl değer atanacağı gösterilmektedir.

```
var
  dizi1: array[2] of string; //static dizi
 {
  dizi1 [0] = 2;
  dizi1 [1] = 5;
  dizi1 [2] = 10;
}
```

Örnek olarak gösterelim.

```
var
  i : Integer;
  dizi1: array[2] of string;
{
  dizi1[0] = 'metin1';
  dizi1[1] = 'metin2';
  for (i = 0 to Length(dizi1)-1)
  {
    ShowMessage(' dizi1['+ IntToStr(i) +'] = '+ dizi1[i]);
  }
}
```

Çok Boyutlu Dinamik Dizi

Çok boyutlu dinamik dizi, çalışma zamanında boyutların değiştirilebildiği ve birden çok boyuta sahip olan bir dizi türüdür. Bu özellik, programın çalışırken boyutun, boyutunun değiştirilebileceği anlamına gelmektedir.

İki boyutlu bir dinamik dizi şu şekilde tanımlanır:

```
var
 dizi1;
```

Bu durumda *dizi1* adında iki boyutlu bir dinamik dizi oluşturulur. Bu dizi başlangıçta boştur ve boyutları çalışma zamanında belirlenebilir. Tanımlama yapıldıktan sonra ölçülerin belirtilmesi gerekir. *VarArrayCreate()* fonksiyonu boyut oluşturmak için kullanılır.

```
function VarArrayCreate(const Bounds: array of Integer; VarType: Integer): Variant;
```

Bu fonksiyon iki parametre almaktadır.

Bounds array of Integer: Bu parametre, oluşturulan çok boyutlu dizinin boyutlarını belirtir. Dinamik bir dizidir ve her boyut için minimum ve maksimum indeks değerlerini içerir. Örneğin, [0, 2, 0, 2] ifadesi iki boyutlu bir dizi oluşturur. Birinci boyutun 0'dan 2'ye kadar indeksleri

vardır, ikinci boyutun da 0'dan 2'ye kadar indeksleri vardır. Bu durumda 3x3'lük bir matris elde edilir.

VarType: Integer: Bu parametre dizideki ögelerin veri tipinin tanımlanması sağlamaktadır. Örneğin *VarType* değeri 3 kullanıldığında dizi içerisindeki ögelerin veri tipi integer olarak tutulacaktır.

Tip	VarType
varEmpty	0
varNull	1
varSmallint	2
varInteger	3
varSingle	4
varDouble	5
varCurrency	6
varDate	7
varError	10
varBoolean	11
varVariant	12
varUnknown	13
varShortInt	16
varByte	17
varWord	18
varLongWord	19
varRecord	36
varString	256

Bu şekilde *VarArrayCreate* işlevi, belirtilen boyutlarda ve istenilen veri türüne sahip çok boyutlu bir dizi oluşturmaktadır.

Aşağıdaki örnekte 6X3'lük bir dizi oluşturulmuş ve değerler atanarak ekran gösterilmiştir.

```
var
 dizi1;
 i,j : Integer;
{
  dizi1 = VarArrayCreate([0, 5, 0, 2], 12);
```

```
  for (i = 0 to 5)
  {
    for (j = 0 to 2)
    {
      dizi1[i, j] = i * 10 + j;
    }
  }

  for (i = 0 to 5)
  {
    for (j = 0 to 2)
    {
      ShowMessage(IntToStr(dizi1[i,j]));
    }
  }
}
```

Standart array tanımlamalarının dışında Clomosy'nin sunmuş olduğu bir yapı bulunmaktadır. Bu yapı *TclArray* yapısıdır. Veri türüne göre farklı tanımlar bulunmaktadır. Bunlar; TclArrayInteger, TclArrayString, TclArrayDouble gibi yapılardır.

Değişken tanımlaması aşağıdaki gibi yapılmalıdır.

```
var

   dizi1 : TClArrayInteger;
```

dizi1 değişkeni tanımlandıktan sonra *TclArrayInteger* tipinde bir nesne oluşturulmalıdır. Bu işlem nesneyi kullanmaya başlamak için gereklidir.

```
dizi1 = TclArrayInteger.Create;
```

Örnek olarak TclArrayInteger tipinde array değişkeni oluşturuldu ve değer atamaları yapıldı. Sonunda bu değerler ekrana gösterildi.

```
var
  dizi1 : TClArrayInteger;
  i : Integer;
{
  dizi1 = TClArrayInteger.Create;
  dizi1.Add(20);
  dizi1.Add(30);
  dizi1.Add(17);
  dizi1.Add(23);
  for (i = 0 to dizi1.Count-1)
```

```
  {
 ShowMessage('Value: '+IntToStr(dizi1.GetItem(i)));
 }
}
```

BÖLÜM 8. E-Posta ve Bildirim

8.1. E-Posta

Clomosy uygulaması aracılığıyla istenen e-posta adresine mesaj gönderebilirsiniz. Bunun için "SendMailNoReplay" kodu kullanılır. Posta gönderirken, e-posta başlığına yazılacak metni "TitleStr" parametresine, e-posta içeriğini "BodyStr" parametresine ve gönderilecek e-posta adresini son parametreye yazmalısınız.

SendMailNoReplay(TitleStr, BodyStr, ReceiverEMail :String):Boolean;

Örnekte example@gmail.com adresine e-posta başlığı 'Clomosy' olan ve mail içeriği 'Clomosy Mail' olan bir mail iletilmektedir.

```
Clomosy.SendMailNoReplay('Clomosy','Clomosy Mail','example@gmail.com');
```

8.2. Bildirim

Bildirimler, bilgi iletmek veya bir şey hakkında uyarı vermek için uygulamaların gönderdiği mesajlardır. Bildirim bileşeni, uygulamanın belirlenmiş bildirim bölgesinde gösterilmek üzere bildirim merkezine gönderdiği mesajdır ve platforma göre değişiklik göstermektedir. Projedeki tüm kullanıcılara gönderilmek istendiğinde "SendNotification" parametresi kullanılmaktadır.

SendNotification(TitleStr, BodyStr, UserGUID :String):Boolean;

Örnek:

```
Clomosy.SendNotification('Clomosy', 'Clomosy Notif', '5F81K4U84J');
```

Belirli kişiler hariç herkese bildirim göndermek için aşağıda bulunan örnekteki gibi gönderilmesini istemediğiniz bir kullanıcı varsa, bu kullanıcıyı seçip diğerlerine bildirim göndermesini etkinleştirebilirsiniz.

SendNotifyAllUsers(TitleStr, BodyStr, WithoutUsers :String):Boolean;

Örnek:

```
Clomosy.SendNotifyAllUsers('Clomosy', 'Clomosy Notif', '5F81K4U84J ,DO5625ARP8');
```

BÖLÜM 9. Formlar

Formlar, Clomosy uygulamalarında kullanıcı ara yüzünü oluşturmanın temel yapı taşlarından biridir. Formlar, görsel nesneleri (controls) ve bu nesnelerin olaylarını (events) içerir. Form üzerinden mobil uygulamalar geliştirilmeye başlanmaktadır. Oluşturulan ilk form projenin ana formu olur. Ana form üzerinde görülebilen veya görünemeyen nesneler yerleştirilerek son kullanıcılar için arayüz tasarlanmaktadır. Masaüstü uygulamalarda olduğu gibi çoklu platform proje yapısında da birden fazla form kullanılabilmektedir. Oluşturulan her form seçili olan cihaz görüntüsünün yükseklik ve genişliğinde boyutlandırılmaktadır.

Clomosy üzerinde kullanılabilecek birden fazla hazır form yapıları bulunmaktadır. Bu sayede proje fikirlerine göre istenilen şekilde proje geliştirme yapılmaktadır. Örnek verilecek olursa bir oyun formu oluşturmak için *TclGameForm* kullanılabilir. Şimdi Clomosy üzerinde kullanılan formlara bakalım.

9.1. TclForm

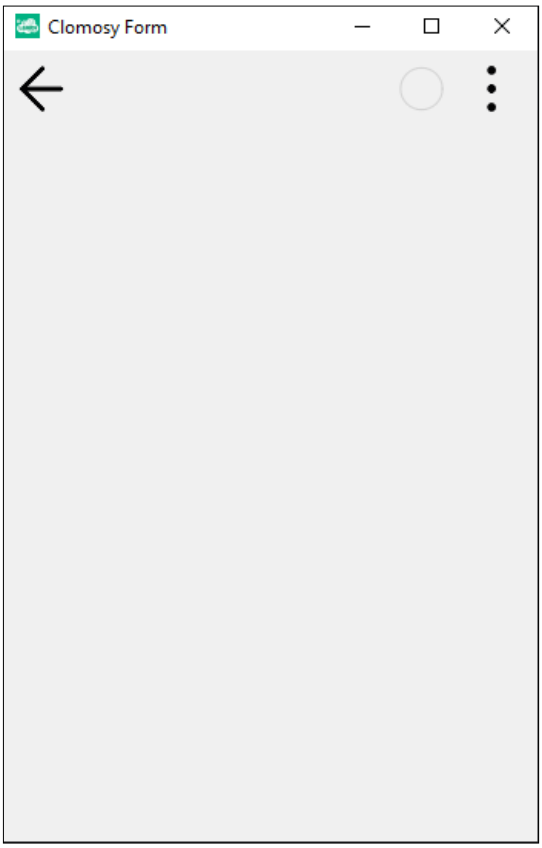
Standart form yapısı oluşturulmak istendiğinde *TclForm* nesnesi oluşturulmaktadır. Bu nesneyi kullanabilmek için *TclForm* sınıfına ait bir nesne değişkeni tanımlamak gerekmektedir.

```
var
  AnaForm:TclForm;
```

Nesne tanımlandıktan sonra oluşturma işlemi *Create* fonksiyonu ile gerçekleştirilmektedir. Formun görünmesi için *Run* fonksiyonu çağrılarak form nesnesi kullanıma hazır hale getirilmektedir.

```
{
  AnaForm = TclForm.Create(Self);
  AnaForm.Run;
}
```

Form oluşturulduktan sonra karşımıza gelecek ekran görüntüsü yan taraftaki gibi olacaktır.



Resim 41

9.2. TclGameForm

Clomosy için özelleştirilmiş bir oyun formudur. Özellikleri arasında animasyon ve ses efektleri gibi özellikler bulunmaktadır. Bu nesneyi kullanabilmek için TclGameForm sınıfına ait bir nesne değişkeni tanımlanır. Nesne tanımlandıktan sonra oluşturma işlemi Create fonksiyonu ile gerçekleştirilir ve formun görünmesi için Run fonksiyonu çağrılarak form nesnesi kullanıma hazır hale getirilmektedir.

```
var
  oyunForm:TclGameForm;
{
  oyunForm = TclGameForm.Create(Self);
  oyunForm.Run;
}
```

URL adresleri alınan dokümanlar, *AddGameAssetFromUrl* parametresi ile asset olarak dosyaya eklenir. Örneğin; arka plan, animasyon görüntüleri ve ses dosyası gibi.

```
oyunForm.AddGameAssetFromUrl('https://www.clomosy.com/game/assets/Tank.png');
```

Görüntüler eklendikten sonra bu görüntüler bir bileşene (TclProImage, TclImage) aktarılabilir. Aktarım yöntemi aşağıdaki gibidir:

```
ImgTank.clSetImage('Tank.png');
```

9.2.1. Animasyon

Animasyonlu resimlerin ve hareketsiz nesnelerin, hareket ediyormuş gibi gösterilmesiyle oluşturulan efekttir. Temelde, bir dizi resim veya nesne çerçevesinin hızlı bir şekilde gösterilmesiyle hareket efekti oluşturulmaktadır.

Animasyonun yüksekliği ve genişliği form üzerinde aşağıdaki gibi ayarlanır.

```
oyunForm.AnimationWidth = 75;
oyunForm.AnimationHeight = 75;
```

Form üzerinde animasyon gecikme süresi *Delay* özelliği ile süresi ise *Duration* ile ayarlanmaktadır.

```
oyunForm.clAnimateBitmap.Delay = 10;

oyunForm.clAnimateBitmap.Duration = 2;
```

Animasyonu ekranda gösterebilmek için *clAnimation* parametresi kullanılmaktadır. Bu parametre 4 değer almaktadır. (x_konumu,y_konumu,tamsayı_değeri, animasyon_document)

Aşağıda rastgele bir konumu ayarlamak için Random() fonksiyonu kullanılmıştır.

```
oyunForm.clAnimation(Random() * oyunForm.clWidth, Random() * oyunForm.clHeight, 20, 'Explosion.png');
```

Aşağıdaki örnekte bir butona tıklandığında patlama efekti verebilen bir resim görüntülenmektedir. Random fonksiyonu ile 5 farklı yerde resim aynı anda görüntülenir ve kaybolur. Örnek BASE dil yapısı ile yazılmıştır.

```
var
  animasyonForm:TclGameForm;
  btnAnimasyon:TclButton;

void BtnPatlatGit;
var
  i:Integer;
{
  animasyonForm.AnimationWidth = 75;
  animasyonForm.AnimationHeight = 75;
  animasyonForm.clAnimateBitmap.AnimationCount =7;
  animasyonForm.clAnimateBitmap.AnimationRowCount=3;
  animasyonForm.clAnimateBitmap.Duration = 2;
  For (i=0 to 5)
  {
    animasyonForm.clAnimateBitmap.Delay = Random()*10;
    animasyonForm.clAnimation(Random() * animasyonForm.clWidth, Random() *
animasyonForm.clHeight, 80, 'Explosion.png');
  }
}

{
  animasyonForm = TclGameForm.Create(Self);

animasyonForm.AddGameAssetFromUrl('https://www.clomosy.com/game/assets/Explosion.png');
```

```
  btnAnimasyon                                                                    =
animasyonForm.AddNewButton(animasyonForm,'btnAnimasyon','PATLAT');
  animasyonForm.AddNewEvent(btnAnimasyon,tbeOnClick,'BtnPatlatGit');
  btnAnimasyon.Align = alBottom;

  animasyonForm.Run;
}
```

9.2.2. Ses Efekti

Uygulamalarınızda bir olaya, uyarı verecek mesajlara veya kullanmak istediğiniz her yere ses ile tetikleme yapmak istendiğinde kullanılmaktadır. İstenilen sesi kullanmak için projeye dahil edilmesi gerekmektedir. Daha sonra *RegisterSound* parametresi ile ses bir değişkene atılmaktadır. Ses efektinin aktif hale getirilmesi için *SoundIsActive* parametresi True olarak ayarlanmalıdır. Son olarak istenilen alanda çalıştırmasını sağlamak için *PlayGameSound* özelliğini kullanmak gerekmektedir. Aşağıdaki kullanımları mevcuttur.

```
anaForm.AddGameAssetFromUrl('https://www.clomosy.com/game/assets/Fire.wav');

SndFire = anaForm.RegisterSound('Fire.wav');

anaForm.SoundIsActive =True;

anaForm.PlayGameSound(SndFire);
```

Bir örnek üzerinde bakalım.

```
var

  sesForm:TclGameForm;

  btnSes:TclButton;

  sesDosyasi:Integer;


void BtnSesCalGit;

{

  sesForm.PlayGameSound(sesDosyasi);

}
```

```
{
  sesForm = TclGameForm.Create(Self);
  sesForm.AddGameAssetFromUrl('https://www.clomosy.com/game/assets/Fire.wav');
  sesDosyasi = sesForm.RegisterSound('Fire.wav');
  sesForm.SoundIsActive=True;

  btnSes = sesForm.AddNewButton(sesForm,'btnSes','SES ÇAL');
  sesForm.AddNewEvent(btnSes,tbeOnClick,'BtnSesCalGit');
  btnSes.Align = alBottom;
  sesForm.Run;
}
```

Yukarıda bir butona tıklandığında ses çalması sağlanmıştır.

9.2.3. Titreşim Efekti

Bir uygulamada, işlemin kullanıcıya uyarısını yapmak için kullanılmaktadır. Clomosy'de bu özelliğin kullanılması için *TclDeviceManager* sınıfı bulunmaktadır. Bu sayede titreşim özelliği aktif hale getirilmektedir. Bu nesneyi kullanabilmek için *TclDeviceManager* sınıfına ait bir nesne değişkeni tanımlanmaktadır.

```
Var

  nTitresim:TclDeviceManager;
```

Nesne tanımlandıktan sonra oluşturma işlemi Create fonksiyonu ile gerçekleştirilmektedir. Ardından titreşim özelliği verebilmek için *Vibrate* fonksiyonuna milisaniye türünden sayı yazılmalıdır ve bu süre kadar cihaz titreşimi aktif hale getirilmektedir.

```
nTitresim = TclDeviceManager.Create;

nTitresim.Vibrate(1000);
```

Bir örnekle kullanımına bakalım. Örnekte bir butona tıklandığında titreşim özelliği aktif hale getirildi.

```
Var
  anaForm:TclGameForm;
  btnTitret:TclButton;
  nTitresim:TclDeviceManager;
Procedure btnTitresimGit;
{
  nTitresim.Vibrate(1000);
}
{
  anaForm = TclGameForm.Create(Self);
  anaForm.SetFormColor('#CBEDD5','#f5e884',clGVertical);
  nTitresim = TclDeviceManager.Create;
  btnTitret= anaForm.AddNewButton(anaForm,'btnTitret','Cihazı titret');
  btnTitret.Align = alBottom;
  btnTitret.Height=100;
  btnTitret.StyledSettings = ssFamily;
  btnTitret.TextSettings.FontColor = clAlphaColor.clHexToColor('#FFFFFF');
  anaForm.AddNewEvent(btnTitret,tbeOnClick,'btnTitresimGit');
  anaForm.Run;
}
```

9.3. TclDrawForm

Clomosy'nin sunmuş olduğu birçok grafik nesnesi ve özelliği içeren özel bir formdur. Çizim işlemleri için kullanılmasının yanı sıra kullanıcıya, çizim, grafik ve diğer görsel etkileşimler sunmaktadır. Örneğin; kullanıcıya çizim araçlarını kullanarak resim çizme veya grafik oluşturma yeteneği kazandırabilir.

TclDrawForm'un özelleştirilebilir özellikleri ve olayları bulunmaktadır. Örneğin kullanıcının fare olaylarına yanıt verebilir ve çizim işlemlerini bu olaylarla senkronize edebilir. Ayrıca çizim için kullanılan grafik nesnelerine doğrudan erişim sağlamaktadır.

Bu nesneyi kullanabilmek için *TclDrawForm* sınıfına ait bir nesne değişkeni tanımlanır. Nesne tanımlandıktan sonra oluşturma işlemi *Create* fonksiyonu ile gerçekleştirilir ve formun görünmesi için *Run* fonksiyonu çağrılarak form nesnesi kullanıma hazır hale getirilir.

```
var

drawForm : TclDrawForm;

{

   drawForm = TclDrawForm.Create(Self);

   drawForm.Run;

}
```

Forma karanlık ve açık bir tema verilebilir. Bunun için *clSetStyle* özelliği kullanılmalıdır. Koyu renk için *DarkSB*, açık renk için *LightSB* kullanılır.

```
drawForm.clSetStyle(drawForm.DarkSB); // drawForm.LightSB
```

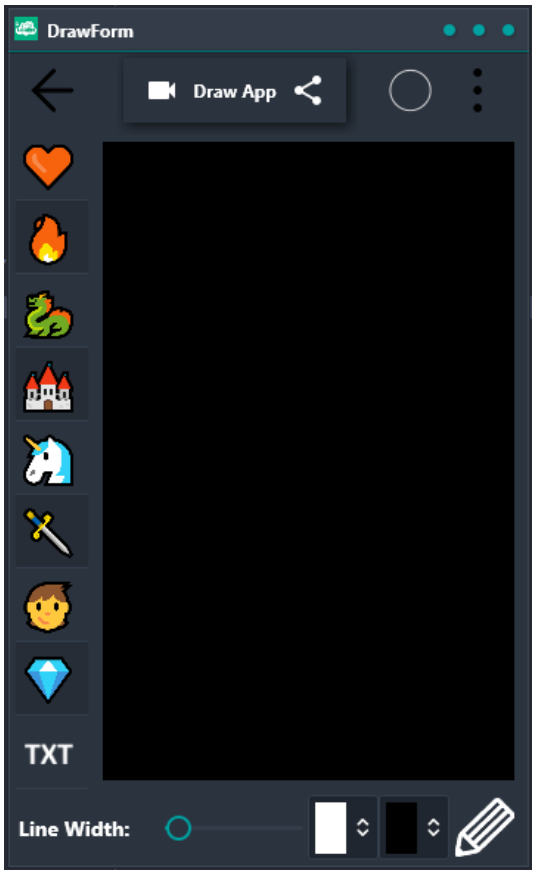
TclDrawForm sınıfının *DrawApp* özelliği bu sınıfın çizim uygulaması olarak kullanılmasını sağlayan bir özelliktir. *DrawApp* özelliği, *TclDrawForm* bileşeninin özelliklerini ve yöntemlerini kullanarak çizim işlemlerini gerçekleştirmek için gerekli araçları ve işlevleri sağlamaktadır.

```
MyDrawForm.DrawApp(True);
```

Sadece formu oluşturalım ve çizim özelliğini de forma ekleyelim.

```
var
drawForm : TclDrawForm;
{
  drawForm = TclDrawForm.Create(Self);
  drawForm.clSetStyle(drawForm.DarkSB); //drawForm.LightSB
  drawForm.DrawApp(True);
  drawForm.Run;
}
```

Form oluşturulduktan sonra karşımıza gelecek ekran görüntüsü aşağıdaki gibi olacaktır.



Resim 42

9.4. TclStyleForm

TclStyleForm, bir formun görünümünü ve davranışını özelleştirmeye yönelik bir mekanizmadır. TclStyleForm, formun arka planını, kenarlık stilini, başlık çubuğunu ve diğer öğelerini özelleştirmek için kullanılmaktadır. Bu sınıf, modern ve şık bir görünüm sağlamak için kullanıcı arayüzü öğelerini ve temalarını yönetmektedir.

Bu nesneyi kullanabilmek için *TclStyleForm* sınıfına ait bir nesne değişkeni tanımlanır. Nesne tanımlandıktan sonra oluşturma işlemi *Create* fonksiyonu ile gerçekleştirilir ve formun görünmesi için *Run* fonksiyonu çağrılarak form nesnesi kullanıma hazır hale getirilmektedir. Tema rengini koyu veya açık olarak ayarlamak için *clSetStyle* özelliği kullanılmaktadır.

```
var
  anaForm :TclStyleForm;
{
  anaForm = TclStyleForm.Create(Self);
  anaForm.clSetStyle(anaForm.DarkSB);
  anaForm.Run;
}
```

9.5. TclGuideForm

Clomosy için özelleştirilmiş bir rehber formudur. Özelleştirilmiş rehber ile tüm verilere ulaşmadan, yalnızca isteğe göre aranan veri döndürmektedir. Diğer bir kullanımda ise bir sayfa aramasından seçilen verilerin bir önceki sayfaya aktarılması istendiğinde "TclGuideForm" bileşeni kullanılmaktadır.

Bu nesneyi kullanabilmek için *TclGuideForm* sınıfına ait bir nesne değişkeni tanımlanır. Nesne tanımlandıktan sonra oluşturma işlemi *Create* fonksiyonu ile gerçekleştirilir ve formun görünmesi için *Run* fonksiyonu çağrılarak form nesnesi kullanıma hazır hale getirilmektedir.

```
var
  guideForm : TclGuideForm;
{
  guideForm = TclGuideForm.Create(Self);
  guideForm.Run;
}
```

Bir rehber formu içerisindeki listeden tıklanan verinin hangi alanı alınmak isteniyorsa *ReturnItemData* fonksiyonu üzerinde tanımlamak gerekir. Bunun yanında ana forma geçiş yapılırken bu veri ana formdaki hangi bileşene döndürülecek ise o bileşeni *ReturnItemObject* fonksiyonunda belirtmek gerekmektedir.

```
guideForm.ReturnItemData = 'MAIN_TEXT';
guideForm.ReturnItemObject = edtListNumber;
```

9.6. Form Özellikler

Her bir form, *TclForm* sınıfından miras alır ve dolayısıyla TclForm'un özelliklerini ve davranışlarını paylaşır. Diğer formlar, yani *TclGameForm, TclDrawForm, TclStyleForm* ve *TclGuideForm*, bu temel form sınıfı olan *TclForm*'dan türetilir. Bu durum, bu formların da *TclForm*'un özelliklerini ve yöntemlerini miras aldığı anlamına gelir. Her form, *TclForm*'un genişletilmiş özelliklerine ve davranışlarına sahip olur, ancak aynı zamanda kendi özel özelliklerini ve davranışlarını da içerebilir.

Örneğin, arka plana renk verme, resim ekleme, formu kapatmak gibi işlemler mevcuttur. Bunların nasıl yapıldığı aşağıda gösterilmektedir.

Arkaplana Renk Verme

SetFormColor fonksiyonu, forma arka plan rengi atamak için kullanılır ve üç parametre almaktadır. İlk iki parametre, kullanılacak renklerin hexadecimal kodlarıdır. Son parametre ise renk paletlerini ayarlamak içindir. Yani tek renk olmasını ya da renk geçişli olmasını sağlamaktadır. Clomosy 'de dört farklı renk palet kullanımı bulunmaktadır.

Özellik	Kullanımı
clGNone	Tekdüze renk için kullanılır.
clGVertical	Dikey olarak degrade olur.
clGHorizontal	Yatay bir eğim oluşturur.
clGCross	Çapraz bir degrade oluşturur.

Kullanımı aşağıdaki gibidir.

```
anaForm.SetFormColor('#CBEDD5','',clGNone);
```

Arkaplana Görsel Ekleme

SetFormBGImage fonksiyonu, forma arka plan görseli eklemek için kullanılmaktadır. İçerisine bir parametre olarak string veri türü alır. Görselin URL adresinin yazılması gerekmektedir.

```
anaForm.SetFormBGImage('https://clomosy.com/theme/SurveyStyle5.png');
```

Form Kapatma

Formların uygulama içerisinde bazı durumlarda kapatılması engellenmek istenmektedir. Örneğin bir sınav oluyor ve sınav bitmeden form kapatılması engellenmek istenebilir. Bu tarz durumlarda Clomosy'nin sağlamış olduğu clCanClose fonksiyonu kullanılmaktadır.

Bu fonksiyonun çalışmasını sağlamak için formun event özelliklerinden *tbeOnFormCloseQuery* kullanılır. Bu event özelliği tanımlandıktan sonra, form kapatılmaya çalışıldığında belirli bir fonksiyon veya prosedür tetiklenir. Bu fonksiyon içinde clCanClose fonksiyonu false olarak ayarlanarak, formun kapatılmasının engellenmesi sağlanmaktadır. Aşağıda kullanımı gösterilmiştir.

```
var
anaForm:TclForm;


void git;
{
  anaForm.clCanClose = false;
}
{
  anaForm = TclForm.Create(Self);
  anaForm.AddNewEvent(anaForm,tbeOnFormCloseQuery,'git');
  anaForm.Run;
}
```

Form Menu ve Çıkış Butonları

Formlar oluşturulduğu anda standart olarak form üzerinde çıkış butonu ve menü butonu bulunmaktadır.

Resim 43

Bunlar form üzerinde istenmiyorsa kapatılabilir. Bu işlemi yapabilmek için form üzerindeki standart butonların isimlerine göre *Visible* değerlerini false yapmak gerekmektedir. Geri butonu bileşeninin ismi *BtnGoBack* ve menü butonu *BtnFormMenu* olarak geçmektedir. Bunların *Visible* değerleri false olduğu anda ekranda görünmemektedir. Kullanımı:

```
TclButton(anaForm.clFindComponent('BtnFormMenu')).Visible = False;

TclButton(anaForm.clFindComponent('BtnGoBack')).Visible = False;
```

Form Yükseklik ve Genişlik Bilgileri

Formun genişlik ve yüksekliğini öğrenebilmek için *clWidth* ve *clHeight* fonksiyonlarını kullanılmaktadır.

```
ShowMessage(anaForm.clHeight);

ShowMessage(anaForm.clWidth);
```

Form Görünürlüğünü Ayarlama

Form gizleme işlemi gerçekleştirmek için *clHide*, formu görünür hale getirmek için *clShow* çağrılmaktadır.

```
anaForm.clHide;

anaForm.clShow;
```

Form Başlığını Değiştirme

Formun başlığını değiştirebilmek için *clSetCaption* prosedürü bulunmaktadır. Bu fonksiyon içerisine string tipinde yeni başlık eklenir. Bu özelliği test edebilmek için Clomosy Windows üzerinde açılmalıdır.

```
anaForm.clSetCaption('newCaption');
```

Form Üzerine Bileşen Ekleme

clGetChild prosedür, Clomosy uygulamasında form içine bir bileşen eklemeyi gerçekleştirir. Bu işlem, form tasarımını oluştururken bileşenlerin dinamik olarak eklenmesini sağlar. Bu fonksiyon bir parametre alır ve bu parametre formun bir alt öğesi olmalıdır.

Kullanımı aşağıda yer almaktadır.

```
var
  anaForm:TcLForm;
  Edit1:TclEdit;

{
  anaForm = TcLForm.Create(Self);

  Edit1 = TclEdit.Create(self);
  anaForm.clGetChild(Edit1);
  Edit1.Align = alCenter;

  anaForm.Run;
}
```

Bileşen Altına Alt Bileşen Ekleme

clGetChildEx, bir form veya uygulamadaki bir bileşenin ebeveynini değiştirmek için tasarlanmış bir prosedür'dür. Prosedür iki parametre almaktadır. İlk parametre, istenilen alt bileşeni ifade eder, ikinci parametre ise ebeveyni ifade etmektedir.

```
Form1.clGetChildEx(child,parent);
```

Aşağıdaki örnekte, formun alt bileşeni olarak layout tanımlanmıştır. Layout değişkeninin alt bileşeni olarak ise label tanımlanmıştır.

```
var
  anaForm:TCLForm;
  Layout1:TclLayout;
  Label1 : TclLabel;

{
  anaForm = TCLForm.Create(Self);

  Layout1 = TclLayout.Create(self);
  anaForm.clGetChild(Layout1);
  Layout1.Align = alCenter;

  Label1 = TclLabel.Create(Self);
  anaForm.clGetChildEx(Label1,Layout1);
  Label1.Caption = 'Alt Bileşen';

  anaForm.Run;
}
```

Kesişen Nesneleri Kontrol Etme

TRObject programlama dilinde kullanılan *clIsIntersects* yöntemi çeşitli geometrik nesneler arasındaki kesişimi kontrol etmek için kullanılan yapıları içermektedir.

Örneğin, iki dikdörtgen arasındaki kesişimi kontrol etmek için kullanılır. Bu yöntem, bir dikdörtgenin başka bir dikdörtgenle kesişip kesişmediğini belirler.

Bu yöntemler, belirli geometrik nesnelerin kesişip kesişmediğini belirleyen bir *Boolean* değer döndürür. Eğer kesişim varsa **True**, yoksa **False** döner. Bu yöntemler sayesinde, programlarımızda çeşitli nesnelerin kesişimini kolayca kontrol edebiliriz.

clIsIntersectsWith

İki nesne arasındaki kesişimi kontrol etmek için kullanılır. Bu metod, bir nesnenin başka bir nesne ile kesişip kesişmediğini belirlemek için kullanılır. Kesiştiklerinde **TRUE** değeri döndürmektedir.

```
MyForm.clIsIntersectsWith(AControl,
BControl:TclControl):Boolean;
```

Aşağıdaki örnekte, bir zamanlayıcı kullanılarak bir Image nesnesinin y pozisyonu artırılarak diğer Image nesnesine doğru yönlendirilir. Eğer bu iki nesne birbiri ile kesişirse, ekrana Label nesnesi üzerinde "Img1 --- Img2 kesişti" mesajı yazdırılır.

```
var
  Form1:TclForm;
  Timer1: TClTimer;
  Img1,Img2: TClProImage;
  Label1: TClProLabel;

void ZamanlayiciAktif;
{
  if (Form1.clIsIntersectsWith(Img1,Img2))
  {
    Timer1.Enabled = False;
    Label1.Visible = True;
    exit;
  }
    Img1.Position.Y = Img1.Position.Y + 30;
}

{
  Form1 = TclForm.Create(Self);

  Label1 = Form1.AddNewProLabel(Form1,'Label1','Img1 --- Img2 kesişti.');
  Label1.Align = AlMostTop;
  Label1.Height = 40;
  Label1.Margins.Left = 30;
  Label1.clProSettings.FontColor = clAlphaColor.clHexToColor('#a11212');
  Label1.clProSettings.FontSize = 24;
  Label1.clProSettings.TextSettings.Font.Style = [fsBold];
  Label1.SetclProSettings(Label1.clProSettings);
  Label1.Visible = False;

  Img1 = Form1.AddNewProImage(Form1,'Img1');
  Img1.Align = AlNone;
  Img1.Height = 50;
  Img1.Width = 50;
  Img1.clProSettings.PictureSource = 'https://clomosy.com/demos/balloon.png';
  Img1.clProSettings.PictureAutoFit = True;
  Img1.SetclProSettings(Img1.clProSettings);
  Img1.Position.X = (Form1.clWidth*0.5);
  Img1.Position.Y = 0;

  Img2 = Form1.AddNewProImage(Form1,'Img2');
  Img2.Align = AlNone;
  Img2.Height = 50;
```

```
  Img2.Width = 50;
  Img2.clProSettings.PictureSource = 'https://clomosy.com/demos/balloon2.png';
  Img2.clProSettings.PictureAutoFit = True;
  Img2.SetclProSettings(Img2.clProSettings);
  Img2.Position.X = (Form1.clWidth*0.5);
  Img2.Position.Y = 300;

  Timer1= Form1.AddNewTimer(Form1,'Timer1',500);
  Timer1.Enabled = True;
  Form1.AddNewEvent(Timer1,tbeOnTimer,'ZamanlayiciAktif');

  Form1.Run;
}
```

clIsIntersectsAny

Bu yöntem, bir nesnenin, verilen bir nesne içindeki herhangi bir nesne ile kesişip kesişmediğini kontrol eder. Bu yöntem, nesnenin başka herhangi bir nesneyle kesişip kesişmediğini hızlıca kontrol etmek için kullanışlıdır. Kesiştiklerinde TRUE değeri döndürmektedir.

```
MyForm.clIsIntersectsAny(AControl, AOwnerControl:TclControl):Boolean;
```

Aşağıdaki örnekte, bir zamanlayıcı kullanılarak bir Image nesnesinin y pozisyonu artırılarak form üzerindeki belirli nesnelerden biriyle çarpıştığı anda, ekrana Label nesnesi üzerinde "Img1 --- Img2 kesişti" mesajı yazdırılır.

```
var
  Form1:TclForm;
  Timer1: TClTimer;
  Img1,Img2,Img3: TClProImage;
  Label1: TClProLabel;


void ZamanlayiciAktif;
{
  if (Form1.clIsIntersectsAny(Img1,Form1)) // Owner'ı Form1 olan tüm nesnelerin Img1 ile
kesiştiğinde geriye true değeri döndürür.
  {
    Timer1.Enabled = False;
    Label1.Visible = True;
    exit;
  }
    Img1.Position.Y = Img1.Position.Y + 30;
}

{
  Form1 = TclForm.Create(Self);
```

```
  Label1 = Form1.AddNewProLabel(Form1,'Label1','Img1 --- Img2 kesişti.');
  Label1.Align = AlMostBottom;
  Label1.Height = 40;
  Label1.Margins.Left = 30;
  Label1.clProSettings.FontColor = clAlphaColor.clHexToColor('#a11212');
  Label1.clProSettings.FontSize = 24;
  Label1.clProSettings.TextSettings.Font.Style = [fsBold];
  Label1.SetclProSettings(Label1.clProSettings);
  Label1.Visible = False;

  Img1 = Form1.AddNewProImage(Form1,'Img1');
  Img1.Align = AlNone;
  Img1.Height = 50;
  Img1.Width = 50;
  Img1.clProSettings.PictureSource = 'https://clomosy.com/demos/balloon.png';
  Img1.clProSettings.PictureAutoFit = True;
  Img1.SetclProSettings(Img1.clProSettings);
  Img1.Position.X = (Form1.clWidth*0.5);
  Img1.Position.Y = 50;

  Img3 = Form1.AddNewProImage(Form1,'Img3');
  Img3.Align = AlNone;
  Img3.Height = 50;
  Img3.Width = 50;
  Img3.clProSettings.PictureSource = 'https://clomosy.com/demos/balloon3.png';
  Img3.clProSettings.PictureAutoFit = True;
  Img3.SetclProSettings(Img3.clProSettings);
  Img3.Position.X = (Form1.clWidth*0.5);
  Img3.Position.Y = 200;

  Img2 = Form1.AddNewProImage(Form1,'Img2');
  Img2.Align = AlNone;
  Img2.Height = 50;
  Img2.Width = 50;
  Img2.clProSettings.PictureSource = 'https://clomosy.com/demos/balloon2.png';
  Img2.clProSettings.PictureAutoFit = True;
  Img2.SetclProSettings(Img2.clProSettings);
  Img2.Position.X = (Form1.clWidth*0.5);
  Img2.Position.Y = 300;

  Timer1= Form1.AddNewTimer(Form1,'Timer1',500);
  Timer1.Enabled = True;
  Form1.AddNewEvent(Timer1,tbeOnTimer,'ZamanlayiciAktif');

  Form1.Run;
}
```

clIsIntersectsList

clIsIntersectsList fonksiyonu, bir kontrol nesnesinin belirli bir kontrole (ya da kontrol listesine) çarpışıp çarpmadığını kontrol eder.

```
MyForm.clIsIntersectsList(AControl, AOwnerControl:TclControl):TclStringList;
```

Bu fonksiyon, *AControl* ve *AOwnerControl* adında iki kontrol nesnesi alır. *AControl*, çarpışmanın kontrol edileceği nesneyi temsil ederken, *AOwnerControl*, *AControl* ile çarpışmanın kontrol edileceği konteyneri temsil eder. Örneğin, *AControl* bir buton ve *AOwnerControl* bir form olabilir.

Fonksiyon, belirli bir kontrolün, belirli bir konteyner içindeki diğer kontrollerle (ya da bir kontrol listesiyle) çarpışıp çarpışmadığını kontrol eder. Eğer çakışma varsa, çakışan kontrollerin adlarını içeren bir *TclStringList* döndürür. Eğer çakışma yoksa, boş bir TclStringList döner.

Aşağıdaki örnekte, bir zamanlayıcı kullanılarak bir Image nesnesinin y pozisyonu artırılarak panel üzerinde bulunan nesneler ile çarpıştığında, çarpışılan nesnelerin adları *TclStringList* üzerine eklenir ve memo üzerinde gösterilir.

```
var
  Form1 : TCLForm;
  Button1 : TClProButton;
  Timer1 : TClTimer;
  Memo1 : TclMemo;
  ListNesne : TclStringList;
  ListSayac : Integer;
  proPnl1 : TclProPanel;
  Img1,Img2: TClProImage;

void ListWrite;
{
  Timer1.Enabled = False;
}
void TimerActive;
var
  i : Integer;
{
  if (Button1.Position.Y > Form1.ClHeight)
  {
    Timer1.Enabled = False;
    exit;
  }
  Button1.Position.Y = Button1.Position.Y + 30;
  Memo1.Visible = True;
```

```
  Memo1.Lines.Clear;
  Memo1.Lines.Add('Object List:');
  ListNesne = Form1.clIsIntersectsList(Button1, proPnl1);
  Form1.clSetCaption(ListNesne.Text);
  ListSayac = ListNesne.Count;
  for (i = 0 to ListSayac - 1)
  {
    Memo1.Lines.Add(Clomosy.StringListItemString(ListNesne,i));
  }
}

{
  Form1 = TCLForm.Create(Self);

  proPnl1 = Form1.AddNewProPanel(Form1,'proPnl1');
  proPnl1.Align = AlClient;
  proPnl1.clProSettings.BackgroundColor = clAlphaColor.clHexToColor('#c0c0c0');
  proPnl1.SetclProSettings(proPnl1.clProSettings);

  Button1 = Form1.AddNewProButton(proPnl1,'Button1','X');
  Button1.Align = AlNone;
  Button1.Height = 50;
  Button1.Width = 50;
  Button1.Position.X = (Form1.clWidth*0.5);
  Button1.Position.Y = 0;

  Img1 = Form1.AddNewProImage(proPnl1,'Img1');
  Img1.Align = AlNone;
  Img1.Height = 50;
  Img1.Width = 50;
  Img1.clProSettings.PictureSource = 'https://clomosy.com/demos/balloon.png';
  Img1.clProSettings.PictureAutoFit = True;
  Img1.SetclProSettings(Img1.clProSettings);
  Img1.Position.X = (Form1.clWidth*0.5);
  Img1.Position.Y = 200;

  Img2 = Form1.AddNewProImage(proPnl1,'Img2');
  Img2.Align = AlNone;
  Img2.Height = 50;
  Img2.Width = 50;
  Img2.clProSettings.PictureSource = 'https://clomosy.com/demos/balloon2.png';
  Img2.clProSettings.PictureAutoFit = True;
  Img2.SetclProSettings(Img2.clProSettings);
  Img2.Position.X = (Form1.clWidth*0.5);
  Img2.Position.Y = 300;

  Memo1 = Form1.AddNewMemo(Form1,'Memo1','');
  Memo1.StyledSettings = ssFamily;
  Memo1.TextSettings.WordWrap = True;
```

```
  Memo1.Align = alCenter;
  Memo1.Visible = False;

  Memo1.Width = 100;
  Memo1.Height = 50;
  Memo1.Margins.Right = 250;
  Memo1.Lines.Add('--------------------');

  ListNesne = Clomosy.StringListNew;
  Timer1= Form1.AddNewTimer(Form1,'Timer1',500);
  Timer1.Enabled = True;

  Form1.AddNewEvent(Timer1,tbeOnTimer,'TimerActive');
  Form1.AddNewEvent(Button1,tbeOnClick,'ListWrite');

  Form1.Run;
}
```

Pencere Boyutu Ayarlama (clSetWindowState)

Clomosy'de *clSetWindowsState*, Windows üzerinde bir pencerenin görüntülenme durumunu belirten bir özelliktir. Bu özellik, bir pencerenin boyutunu, konumunu ve durumunu yönetmeye yardımcı olmaktadır.

clSetWindowsState özelliği, genellikle TclForm sınıfı (veya TclStyleForm gibi diğer formlar) için kullanılır. Bu özellik, aşağıdaki değerlerden birine sahip olabilir:

Özellik	**Amacı**
fwsMinimized	Form penceresini simge durumuna küçültür.
fwsNormal	Form penceresini normal pencere görünümü yapar.
fwsMaximized	Form penceresini tam ekran durumuna getirir.

Örnek kullanımı:

```
MyForm.clSetWindowState(fwsMaximized)
```

Bu özellik, bir pencerenin başlangıç durumunu veya çalışma zamanında durumunu belirlemek için kullanılır. Örneğin, bir formun başlangıç durumunu *fwsMaximized* olarak ayarlayarak, uygulama başladığında pencerenin tam ekran olarak açılması sağlanılır.

Aşağıdaki örnekte 3 adet buton oluşturuldu ve bu butonlara tıklandığında, tıklanan butona göre pencere boyutu değişmektedir.

```
var
  Form1:TCLForm;
  normalButton,maximizeButton,minimizeBtn : TclButton;
void butonaTiklandiginda;
var
  senderBtn:TclButton;
{
  senderBtn = TclButton(Form1.ClSender);
  case senderBtn.ClTagInt of
  {
    0:Form1.clSetWindowState(fwsNormal);
    1:Form1.clSetWindowState(fwsMaximized);
    2:Form1.clSetWindowState(fwsMinimized);
  }
}
{
  Form1 = TCLForm.Create(Self);

  normalButton = Form1.AddNewButton(Form1,'normalButton','Normal');
  NormalButton.Align = alTop;
  NormalButton.Margins.Bottom = 50;
  Form1.AddNewEvent(normalButton,tbeOnClick,'butonaTiklandiginda');

  maximizeButton = Form1.AddNewButton(Form1,'maximizeButton','Maximize');
  maximizeButton.ClTagInt = 1;
  maximizeButton.Margins.Bottom = 50;
  maximizeButton.Align = alTop;
  Form1.AddNewEvent(maximizeButton,tbeOnClick,'butonaTiklandiginda');

  minimizeBtn = Form1.AddNewButton(Form1,'minimizeBtn','Minimize');
  minimizeBtn.ClTagInt = 2;
  minimizeBtn.Align = alTop;
  Form1.AddNewEvent(minimizeBtn,tbeOnClick,'butonaTiklandiginda');

  Form1.Run;
}
```

Pencere Konumunu Ayarlama (clSetPosition)

Clomosy'de *clSetPosition*, Windows üzerinde bir pencerenin ekranda nasıl konumlandırılacağını belirleyen bir dizi seçenek içerir. Bu seçenekler, formun boyutu ve pozisyonu üzerinde kontrol sağlar. TFormPosition ile belirli bir formun açıldığında nerede ve hangi boyutta olacağını ayarlayabilirsiniz.

clSetPosition özelliği, genellikle TclForm sınıfı (veya TclStyleForm gibi diğer formlar) için kullanılır. Bu özellik, aşağıdaki değerlerden birine sahip olabilir:

Özellik	Amacı
fpDesigned	Formun konumu ve boyutu, tasarım zamanında ayarlanan değerlere göre belirlenir. Uygulama açıldığında ekranın sol üst kısmında görüntülenir.
fpDefault	Form, işletim sisteminin varsayılan kurallarına göre yerleştirilir ve boyutlandırılır.
fpDefaultPosOnly	Formun pozisyonu işletim sistemine göre ayarlanır, ancak boyutu tasarım zamanında belirlenen boyut olarak kalır.
fpDefaultSizeOnly	Formun boyutu işletim sistemine göre ayarlanır, ancak pozisyonu tasarım zamanında belirlenen pozisyon olarak kalır. Uygulama açıldığında ekranın sol üst kısmında görüntülenir.
fpScreenCenter	Form, ekranın merkezine yerleştirilir.
fpDesktopCenter	Form, masaüstünün merkezine yerleştirilir. (Çoklu monitör sistemlerinde geçerlidir.)
fpMainFormCenter	Form, ana formun merkezine yerleştirilir.
fpOwnerFormCenter	Form, sahip formun merkezine yerleştirilir (eğer bir sahibi varsa).

Örnek kullanımı:

```
MyForm.clSetPosition(fpDefault)
```

Yukarıdaki örnekte, MyForm.*clSetPosition* adında bir yöntem bulunuyor ve bu yöntemin parametre olarak TclFormPosition türünden bir değer aldığı görülüyor. Bu metodun nasıl çalıştığını belirlemek için yöntemin tanımına bakmak gerekir.

Klavye Girişini Yakalama

Clomosy'de *clSenderKeyChar*, bir klavye olayı işlendiğinde, bu olayı tetikleyen bileşeni (genellikle bir klavye olayı bir forma veya bir düğmeye bağlıdır) temsil eden bir değeri içerir. Bu değer, kullanıcının klavyeden girdiği karakteri veya tuşun karakter değerini içermektedir.

Bu özellikten geriye bir değer alabilmek için *tbeOnKeyUp* ve *tbeOnKeyDown* olayları (event) kullanılır.

tbeOnKeyDown klavye üzerinde bir tuşa basıldığı anda tetiklenirken *tbeOnKeyUp* klavyede basılmış bir tuş bırakıldığı zaman tetiklenir. Klavyeden alınan girdi bilgisine göre işlem yapabilmek için formun *clSenderKeyChar* özelliğini kullanılır. Bu özellik ile klavyeden basılan ya da çekilen tuş bilgisi decimal değerde tutulmaktadır. Eğer klavye tuşlarının decimal karşılığını bilmiyorsanız 'Key Code Table' araması ile rahatlıkla decimal kodlarını bulabilirsiniz.

Aşağıdaki örnekte basılan tuşa göre gerçekleşecek olay ile hangi tuşa basıldığı tespit edilerek, basılan tuşla ilgili mesaj gösteren bir uygulama yapılmıştır.

```
var
  Form1:TCLForm;

void KeyDown
{
  if (Form1.clSenderKeyChar == 32)
  {
    ShowMessage('Space Tuşuna Basıldı');
  }
  else if (Form1.clSenderKeyChar == 119)
  {
    ShowMessage('W Tuşuna Basıldı');
  }
}

void KeyUp
{
  if (Form1.clSenderKeyChar == 97)
  {
    ShowMessage('A Tuşundan Çekildi');
  }
  else if (Form1.clSenderKeyChar == 100)
  {
    ShowMessage('D Tuşundan Çekildi');
  }
}

{
  Form1 = TCLForm.Create(Self);
  Form1.AddNewEvent(Form1, tbeOnKeyDown, 'KeyDown');
  Form1.AddNewEvent(Form1, tbeOnKeyUp, 'KeyUp')
  Form1.Run;
}
```

Projeye Dosyasına Görsel Ekleme

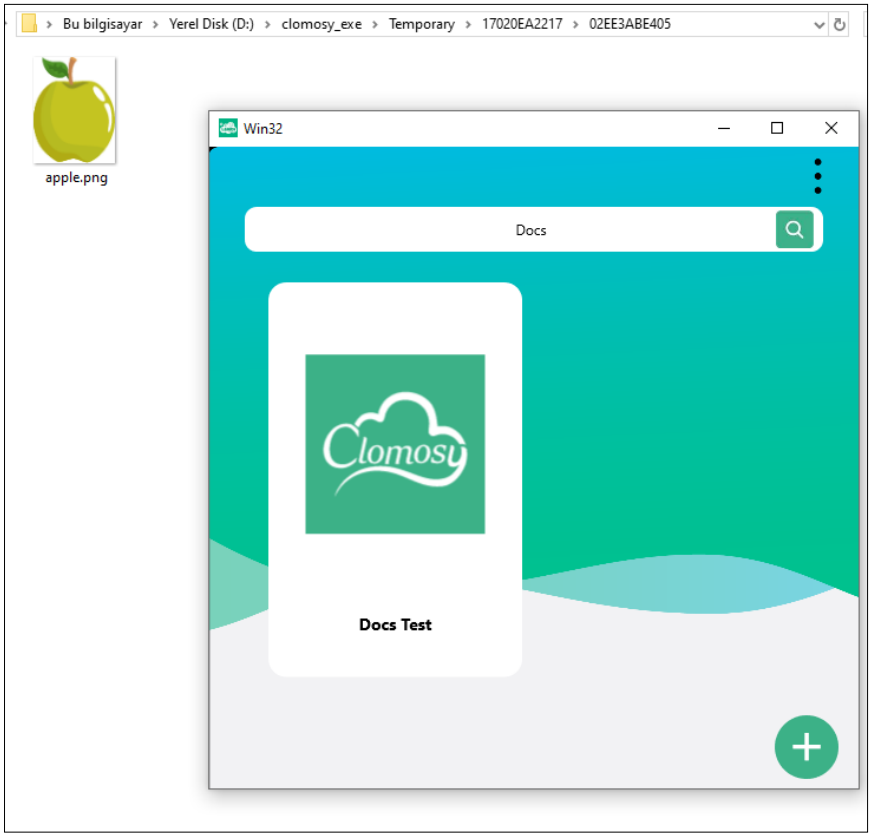
AddAssetFromURL, Clomosy'de bir URL'den bir varlığı (asset) indirip uygulamanıza dâhil etmek için kullanılan bir işlevdir. Bu işlev bir dosyayı (örneğin, bir resim dosyası) internetten indirip kullanılan projenin dosya sistemine kaydetmek için kullanılır.

Aşağıdaki örnekte url adresi belirtilen görseli projenin dosya yoluna görsel olarak kaydetmektedir.

i	Görselin dosyaya kaydedildiğini kontrol etmek için projede *Clomosy.AppFilesPath* yazarak dosya yolunun uzantısı öğrenilir. (Windows üzerinde dosyaya bakın.)

```
var
  Form1 : TCLForm;
 {
  Form1 = TCLForm.Create(Self);
  Form1.AddAssetFromUrl('https://clomosy.com/demos/apple.png');
  //ShowMessage(Clomosy.AppFilesPath);
  Form1.Run;
 }
```

Dosyaya kaydedilip çalıştırıldığında ve dosya kontrol edildiğinde aşağıdaki gibi bir görsel dosyaya eklendiği görülür.



9.7. Event (Olay)

Olay, bileşenin çalışma veya tasarım anında kullanıcının etkisi ile ortaya çıkardığı davranışlardır. Her davranış bir sonuç bildirmektedir. Bir bileşen üzerinde gerçekleşen tüm aktivite sonucunda gerçekleşen bildirimler **event** (olay) olarak adlandırılmaktadır.

Bir bileşene event özelliği verebilmek için form üzerinde *AddNewEvent* fonksiyonunun kullanılması gerekmektedir.

AddNewEvent fonksiyonu üç parametre almaktadır. İlk parametre event kullanılacak bileşeni temsil etmektedir. İkinci parametre event özelliklerinden hangisinin kullanılması gerektiğini, üçüncü parametre ise olay tetiklendiğinde yapılacak olan işlemi ifade etmektedir. Burada prosedüre veya fonksiyona yönlendirme yapılabilmektedir.

Aşağıda bir örneği yapılmıştır. Bir *AraBtn* buton bileşenine tıklandığında yani *tbeOnClick* event özelliği kullanıldığında *BtnGit* prosedürüne yönlendirilmektedir.

```
anaForm.AddNewEvent(AraBtn,tbeOnClick,'BtnGit');
```

Yazılım geliştiriciler olarak bileşenin hangi davranışında bir iş gerçekleştirecek isek o olayda gerekli kodları yazarız. Örneğin mouse ile bir alana geldiğinde bir uyarı mesajı verdirmek için o alanda bulunan bileşenin *OnMouseMove* olayını kullanırız.

Form üzerinde gerçekleşen olayların bir kısmı son kullanıcının etkisiyle, bir kısmı form doğası gereği, diğer bir kısmı ise işletim sistemine özgü bazı davranışların etkisiyle ortaya çıkmaktadır.

Aşağıda event kullanılan bazı özellikler bulunmaktadır.

Özellik	Kullanım Amacı
tbeOnClick	Bu olay, bir bileşene (genellikle buton) tıklama eylemi tetiklendiğinde meydana gelir. En temel olaydır ve kullanıcının bir eylem başlatmasına olanak tanır.
tbeOnEnter	Bir bileşene odaklanıldığında (klavyeyle veya mobil üzerinden o bileşen seçildiğinde, fareyle tıklandığında) tetiklenir. Bu özellikle klavyede gezinme ve etkileşim için önemlidir.
tbeOnExit	Genellikle bir giriş alanından çıkarken (örneğin bir bileşenden diğerine geçiş yaparken) gerçekleştirilmesi gereken eylemler için kullanılır.
tbeOnFormClose	Bu, form kapatıldığında tetiklenecek bir olaydır.
tbeOnFormCloseQuery	Form kapatıldığında bazı olaylar için tetiklenecek bir olaydır.
tbeOnFormHide	Form gizlendiğinde tetiklenecek bir olaydır.

tbeOnFormShow	Form görünür hale geldiğinde tetiklenir.
tbeOnGesture	Bu, TclStringGrid bileşenine tıkladığınızda tetiklenen olaydır. Belirli bir bileşen (görüntü veya form gibi) üzerinde belirli bir hareket veya eylemin tetiklediği olayı ifade eder.
tbeOnMouseDown	Bir bileşene (bir düğme veya resim gibi) fare ile tıklandığında tetiklenen bir olaydır. Bu olay fare tuşuna basıldığında meydana gelir.
tbeOnMouseUp	Bir bileşenin fareyle tıklandıktan sonra serbest bırakılmasıyla tetiklenen bir olaydır.
tbeOnMouseMove	Fare işaretçisini bir bileşenin üzerine getirdiğinizde tetiklenen bir olaydır.
tbeOnKeyDown	Klavyede bir tuşa basıldığı an tetiklenen olaydır.
tbeOnKeyUp	Klavyede bir tuşa basma işlemi sonlandığı an tetiklenen olaydır.

Daha fazla TclBasisEvent özelliklerine erişmek için https://www.docs.clomosy.com sitesini ziyaret edebilirsiniz.

BÖLÜM 10. Bileşenler

Kullanıcı arayüzlerinin tasarımında veya işlevsel ihtiyaçlarda kullanılan, birçok özelliği, metodu veya tipi içeren ve tasarım aşamasında oluşum hiyerarşisini gösteren nesnelere "bileşen" denilmektedir. Bileşenler component veya nesne olarak da adlandırılmaktadır. Örneğin button, edit, label, panel ve bunlar gibi birden fazla bileşen bulunmaktadır. Bileşenin sahip olduğu özellik ve metodların tümüne **bileşenin üyeleri** adı verilmektedir. Clomosy yazılım literatüründe özellikler **property**, metodlar **function** veya **void (prosedür)** adı ile anılmaktadır.

Uygulama çalışma anında, bileşenlerin görünürlük durumlarına göre **Visual** ve **Non-Visual** olmak üzere iki gruba ayrılmaktadır. Visual bileşenler, uygulama çalışma anında son kullanıcılar tarafından görünen, non-visual bileşenler ise görünmeyen bileşenlerdir. Bunlara örnek olarak visual için button, non-visual için ise timer bileşenleri örnek verilebilir.

Clomosy'de 'standart bileşenler' ve 'profesyonel bileşenler' olarak iki tip bileşen bulunmaktadır. Kullanımları farklıdır. Standart bileşene örnek olarak *TclButton*, profesyonel bileşen için *TclProButton'u* örnek verebiliriz. Profesyonel bileşenler için *TclPro* diyerek başlanmaktadır. Bu iki yapının örnek kullanımlarından bahsedelim. Mesela buton üzerinde genişlik ve yükseklik ayarlaması yapalım.

Standart yapıda kullanım;

```
TestBtn.Width = 100;
TestBtn.Height = 100;
```

Profesyonel bileşenlerde *clProSettings* (Tercih edilen) veya *SetupComponent* kullanılarak özellikler verilmektedir. Bunun haricinde standart buton tanımlaması gibi de özelliklere değer atamaları yapılmaktadır. Fakat standart yapılarda kullanılmayan özellikler bu yapılar ile tanımlanmaktadır. Mesela genişlik ve yükseklik verileceği zaman standart yapı ile kullanılabilir ama arka plan rengi verme gibi durumlarda *clProSettings* (önerilen) veya *SetupComponent* kullanılmaktadır.

clProSettings Kullanımı;

Standart bileşenlerin özelliklerinin dışında gelişmiş özelliklerin kullanılması için alternatif bir kullanımıdır.

Örneğin;

Bir buton bileşenine yazı rengi ve kalınlığı vermek için aşağıdaki gibi tanımlanır.

```
Button1.clProSettings.FontColor = clAlphaColor.clHexToColor('#ff00ac');
Button1.clProSettings.TextSettings.Font.Style = [fsBold];
Button1.SetclProSettings(Button1.clProSettings);
```

SetupComponent Kullanımı;

Bu fonksiyon iki parametre almaktadır. İlk parametreye özellik verilecek bileşen adı yazılır, ikinci parametreye json yapısı içerisinde bileşene özellikler verilmektedir.

Örneğin;

Bir buton bileşenine yazı rengi ve kalınlığı vermek için aşağıdaki gibi tanımlanır.

```
clComponent.SetupComponent(Button1,'{ "TextColor":"#f5428d", "TextBold":"yes"}');
```

clProSettings (Tercih edilen) veya *SetupComponent* tanımlamalarına verilen diğer özellikleri ve kullanımlarını incelemek için https://www.docs.clomosy.com/Pro_Object_Properties sayfasına bakınız.

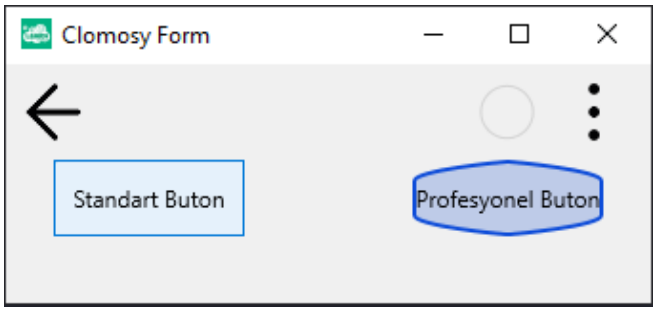
Konu kapsamında kullanılan bileşen tanımlamaları amaçları detaylı olarak anlatılacaktır. Anlatılacak bu bileşenlerin ayrıntılı kullanımı ve örneklerine https://www.docs.clomosy.com/Components sayfasından erişebilirsiniz.

10.1. Buton Tipi Bileşenler

Butonlar kullanıcı arayüzlerinin vazgeçilmez bileşenleridir. Bir işi gerçekleştirmek için kullanılmaktadır. Genellikle *tbeOnClick* olayına kod yazılarak fonksiyonel hale getirilmektedir. Şekil ve amacına göre iki buton çeşidi bulunmaktadır. Bunlar, *TclButton* ve *TclProButton* olarak geçmektedir. Butonlar uygulama arayüzünde bulundukları modül veya fonksiyonlara

göre seçilmektedir. Örneğin; bir buton üzerinden görsel istendiğinde *TclProButton* bileşeninde görsel ekleme olduğu için bu buton tercih edilmektedir.

Aşağıda Clomosy üzerinde kullanılan butonlar gösterilmiştir. *TclProButton* görsellik açısından daha çok tercih edilmektedir. Arka plan, çerçeve ve metin rengi verme gibi özelliklere sahiptir. Standart bir projede TclButton kullanılabilir.



Resim 44

10.1.1. TclButton

TclButton, uygulamalarda kullanılmak üzere genel amaçlı bir butondur. Butona tıklandığında, bir işlev, prosedür veya diğer eylemler etkinleştirilir veya bu işlevler yönlendirilmektedir.

Bir butonu oluşturmak için *AddNewButton* fonksiyonu kullanılmaktadır. Bu işlev üç parametre almaktadır. İlk parametre tanımlanacak olan butonun hangi bileşen içerisinde konumlanacağını belirlemektedir. İkinci parametre, tanımlanan butonun ismini temsil etmektedir. Üçüncü parametre ise butona başlık vermektedir.

Örnekte bir buton oluşturalım. Butonu, form içerisine tanımlayacağız. Bunun için ilk parametre oluşturulan formun adını almalıdır. Buton değişkenine *testBtn* adını verelim. Son olarak buton üzerinde bulunacak başlıktır. Başlığını Tıkla olarak yazalım.

```
var
  anaForm:TclForm;
  testBtn : TclButton;
{
  anaForm = TclForm.Create(Self);

  testBtn = anaForm.AddNewButton(anaForm,'testBtn','Tıkla');

  anaForm.Run;
}
```

10.1.2. TclProButton

Standart buton bileşeninin aksine, üzerine resim ekleyebilme özelliğine sahiptir. Bunun dışında, standart buton kullanıldığında uygulama çalıştırıldığı anda butonda tıklama efekti

bulunmamaktadır. Bu butonu oluşturmak için *AddNewProButton* fonksiyonu kullanılmaktadır. Tanımlama mantığı standart buton ile aynıdır.

```
var
  anaForm:TclForm;
  testBtn : TclProButton;
{
  anaForm = TclForm.Create(Self);
  testBtn = anaForm.AddNewProButton(anaForm,'testBtn','Tıkla');
  anaForm.Run;
}
```

10.2. Seçim Tipi Bileşenler

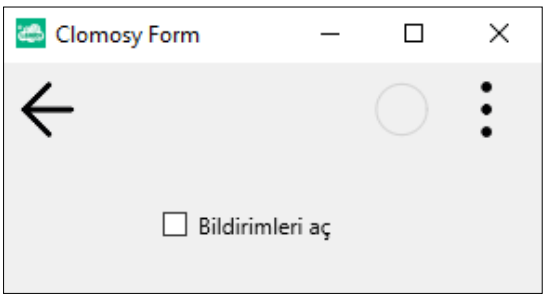
Bir veya daha fazla seçenek sonucuna göre işlem yapılıcaksa seçim tipi bileşenler kullanılmaktadır. Bu tip bileşenlere *TclCheckBox, TclRadioButton* örnek verilebilir.

10.2.1. TclCheckBox

Birden fazla veya tek seçim yapmak için kullanılmaktadır. Bir checkbox oluşturmak için *AddNewCheckBox* fonksiyonu kullanılır. Bu işlev üç parametre almaktadır. İlk parametre tanımlanacak olan checkbox'ın hangi bileşen içerisinde konumlanacağını belirlemektedir. İkinci parametre, tanımlanan checkbox'ın ismini temsil etmektedir. Üçüncü parametre ise başlık vermektedir.

```
var
  anaForm:TclForm;
  testCheckBox : TClCheckBox;
{
  anaForm = TclForm.Create(Self);
  testCheckBox = anaForm.AddNewCheckBox(anaForm,'testCheckBox','Bildirimleri aç');
  testCheckBox.Align = AlCenter;
  anaForm.Run;
}
```

Aşağıda örnek bir gösterimi bulunmaktadır.



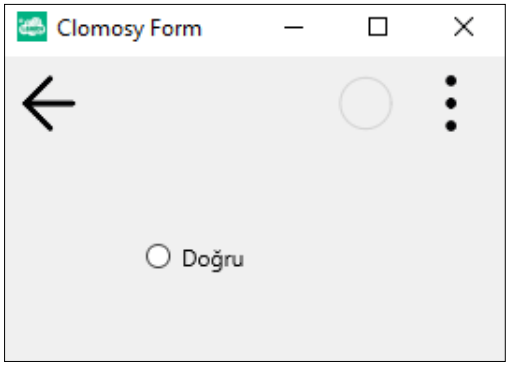
Resim 45

10.2.2. TclRadioButton

Tek seferde bir gruptan yalnızca tek seçim yapılması gereken bir türdür. Bir radiobutton oluşturmak için *AddNewRadioButton* fonksiyonu kullanılmaktadır. Bu işlev üç parametre almaktadır. İlk parametre, RadioButton'un hangi bileşen içerisinde konumlanacağını belirlemektedir. İkinci parametre, tanımlanan RadioButton'un ismini temsil etmektedir. Üçüncü parametre ise başlık vermektedir.

```
var
  anaForm:TclForm;
  testRadio : TClRadioButton;
{
  anaForm = TclForm.Create(Self);

  testRadio = anaForm.AddNewRadioButton(anaForm,'testRadio','Doğru');

  anaForm.Run;
}
```

Aşağıda örnek bir gösterimi yer almaktadır.



Resim 46

10.3. Liste Tipi Bileşenler

Genellikle birden fazla kayıt listelemek için kullanılan bileşenlerdir. Belli bir grup kayıt yalnızca kullanıcılara göstermek amaçlı veya listeden kayıt seçmek amaçlı kullanılmaktadır. Bunlardan bazıları 'ComboBox' tek seçim amaçlı, 'ListView', 'StringGrid' gibi bileşenler ise hem listeleme hem de seçim amaçlıdır. ComboBox dışında diğer bileşenler ile bir veya daha fazla seçim, seçilmiş kayıtlar üzerinde toplu işlem yapılmaktadır.

10.3.1. TclComboBox

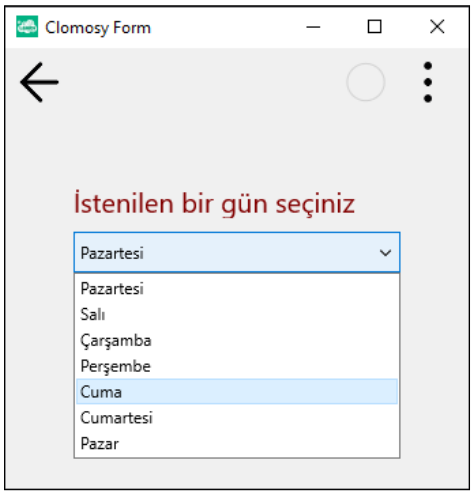
ComboBox liste şeklinde açılan bir kutudur. Veri listeleme, veri seçimi gibi olaylarda kullanılmaktadır. Görünümü TclEdit nesnesine yakındır. Sağ tarafta, aşağı doğru açılmasını

sağlayacak bir ok butonu bulunmaktadır. Bu sayede kilit açma butonuna basıldığında içerisinde listelenen veriler arasından seçim yapma imkânı sağlamaktadır.

Bir ComboBox oluşturmak için *AddNewComboBox* fonksiyonu kullanılmaktadır. Bu işlev iki parametre almaktadır. İlk parametre, tanımlanacak olan comboBox hangi bileşen içerisinde konumlanacağını belirlemektedir. İkinci parametre, tanımlanan ComboBox ismini temsil etmektedir.

```
var
  anaForm:TclForm;
  testCombo : TClComboBox;
{
  anaForm = TclForm.Create(Self);

  testCombo = anaForm.AddNewComboBox(anaForm,'testCombo');
  testCombo.AddItem('Test 1','01');
  anaForm.Run;
}
```

Aşağıda örnek bir gösterimi bulunmaktadır.



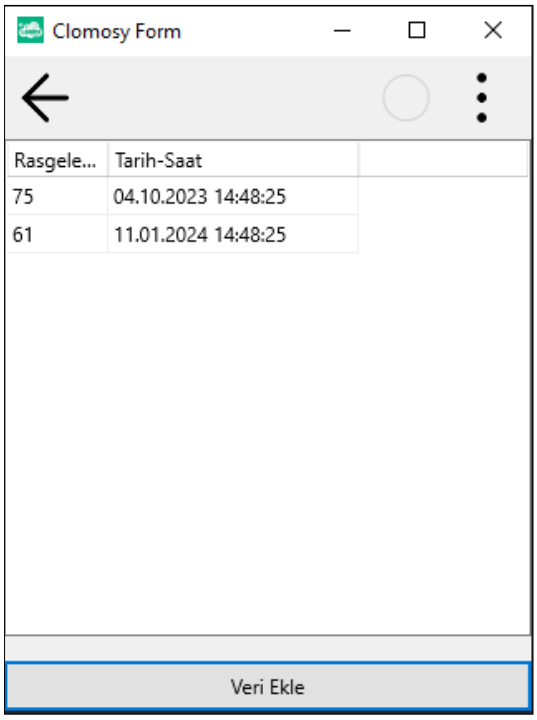
Resim 47

10.3.2. TclStringGrid

Dizelerin ve ilgili nesnelerin işlenmesini basitleştirmek için tasarlanmış bir ızgara kontrolünü temsil etmektedir. Metin verilerini tablo biçiminde sunmak için forma bir *TclStringGrid* eklenir. Bir TclStringGrid nesnesini oluşturmak için *AddNewStringGrid* fonksiyonu kullanılmaktadır. Bu işlev iki parametre almaktadır. İlk parametre tanımlanacak olan nesnenin hangi bileşen içerisinde konumlanacağını, ikinci parametre, nesnenin ismini temsil eder.

```
var
  GridForm:TclForm;
  anaGrid:TclStringGrid;
{
  GridForm = TClForm.Create(Self);

  anaGrid = GridForm.AddNewStringGrid(GridForm, 'anaGrid');

  GridForm.Run;
}
```

Aşağıda örnek bir gösterimi yer almaktadır.



Resim 48

10.3.3. TclListView

TclListView, çeşitli tiplerdeki ögeleri tutmak ve sunmak için kullanabileceğiniz bir liste görünümü bileşendir. TclListView, hızlı ve sorunsuz kaydırma için optimize edilmiş bir liste ögeler koleksiyonunu görüntülemektedir. Ayrıca veriler, veri tabanından çekilmek istendiğinde, bu bileşen kullanılmaktadır. Bir ListView, AddNewListView fonksiyonu ile oluşturulmaktadır. Bu işlev iki parametre almaktadır. İlk parametre, tanımlanacak olan nesnenin hangi bileşen içerisinde konumlanacağını, ikinci parametre nesnenin ismini temsil etmektedir.

```
Var
  anaForm:TclForm;
  testListview : TClListView;
{
  anaForm = TclForm.Create(Self);
  testListview = anaForm.AddNewListView(anaForm,'testListview');
  anaForm.Run;
 }
```

ListView içerisine bir veri kümesinden veri alınmak istendiğinde clLoadListViewDataFromDataset fonksiyonu kullanılmaktadır. Bu fonksiyon, yalnızca json veri yapısını kullanmaz. Bu fonksiyon, dataset nesnesinden gelen verileri, TclListView kontrolündeki belirli sütunlara yerleştirmektedir.

Dataset nesnesi, bir SQL sorgusu sonucunda elde edilen verileri içeren veri kümesidir. Bu veri kümesi içindeki veriler, *TclListView*'ın belirli sütunlarına *clLoadListViewDataFromDataset* fonksiyonu ile *TclListView* verisine dönüştürülerek listeye aktarılmaktadır. clLoadListViewDataFromDataset fonksiyonu, bu veri aktarımını yönetmektedir.

```
testListview.clLoadListViewDataFromDataset(foodListQuery);
```

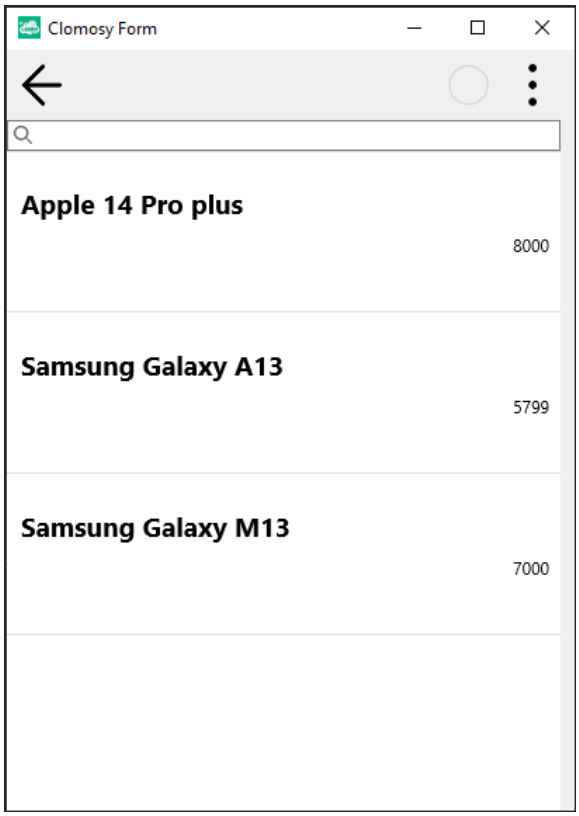
ListView içerisine verileri ekleyebilmek için ListView alanlarının isimlerine göre gelen verileri adlandırmak gerekmektedir. Bu durumda ListView alanlarını bilmek gerekir. Aşağıda ListView içerisinde bulunan alanlar görülmektedir.

Resim 49

Örneğin; veri tabanından çekilip ListView içerisine atılmak isteniyor. Bunun için veriler gelirken tabloda alan isimleri ListView alan isimleri olmalıdır. Aşağıda *Urunler* tablosundan *urunId,urunAdi, urunUcreti* alınmak istenmektedir ve görüldüğü gibi ListView eklenirken alan adları değiştirilmektedir.

```
void listViewVeriEkle;
Var
  urunLıstQry : TClSqlQuery;
{
  urunLıstQry = TClSqlQuery.Create(nil);
  try
    urunLıstQry.Connection = Clomosy.DBSQLServerConnection;
    urunLıstQry.SQL.Text = 'SELECT urunId as RECORD_GUID, urunAdi as
MAIN_TEXT, urunUcreti as SIDE_TEXT_BOTTOM from Products';
    urunLıstQry.Open;
    if (urunLıstQry.Found)
    {
      testListview.clLoadListViewDataFromDataset(urunLıstQry);
    }
  finally
    urunLıstQry.Close;
    urunLıstQry.Free;
  }
}
```

Aşağıda örnek bir gösterimi bulunmaktadır.



Resim 50

10.3.4. TClProListView

Profesyonel ListView'in kullanım amacı, *TclListView* kullanım amacıyla aynıdır. Standart TclListView parametre özelliklerinin dışında, arka plan ve kenarlık rengi oluşturma gibi özelliklere sahiptir. TclProListView'deki bileşenlerin görünümünü de kendimiz tasarlayabiliriz. Standart TclListview nesne tanımı yapıldığında arama çubuğu otomatik olarak üst kısımda tanımlanmaktadır. TclProListView nesne tanımlaması yapıldığında arama çubuğu gelmemektedir. Bunun için Tcl*ProSearchEdit* bileşenini tanımlamamız gerekmektedir.

Nesne alanlarına veriler eklenebilmesi için başka bir bileşen olan Tcl*ProListViewDesignerPanel* eklenmelidir ve bu şekilde birden fazla listeye ögeler eklenmesi sağlanabilir. Bu bölümde TclProListView nasıl tanımlanır görelim. Tcl*ProListViewDesignerPanel* bölümünde veri ekleme gösterilecektir.

```
Var
  anaForm:TclForm;
  testListview : TClProListView;
{
  anaForm = TclForm.Create(Self);
  testListview = anaForm.AddNewProListView(anaForm,'testListview');
  anaForm.Run;
}
```

Liste tipi *ListType* özelliği ile ayarlanmaktadır. İki şekilde tanımlama yapılmaktadır. Listedeki verilerin yatay olması için *horizontal*, dikey olabilmesi için *vertical* yazılmaktadır. Liste içerisindeki nesnelerin arasındaki boşluk *ItemSpace* ile belirlenmektedir.

```
testListview.ListType = 'horizontal';
testListview.Properties.ItemSpace = 10;
```

10.3.5. TClProListViewDesignerPanel

Listview'de tanımlanabilen, eleman ekleyip tasarlayabileceğimiz bir bileşendir. Tanımlaması aşağıdaki gibidir. TclProListViewDesignerPanel oluşturmak için *AddNewProListViewDesignerPanel* kullanılmaktadır. Burada birinci parametreye oluşturulan TclProListView bileşeninin adı yazılmaktadır. Bu bileşen bir TclProListView içerisinde kullanıldığı için bu şekilde tanımlanmaktadır.

```
var
   anaForm : TclForm;
   listView1 : TClProListView;
   designerPanel1 : TClProListViewDesignerPanel;
{
  anaForm = TclForm.Create(Self);

  listView1 = anaForm.AddNewProListView(anaForm,'listView1');
  listView1.Align = AlClient;
  listView1.clProSettings.ViewType = lvIcon;
  listView1.clProSettings.BorderColor = clAlphaColor.clHexToColor('#0850c4');
  listView1.clProSettings.BorderWidth = 3;
  listView1.SetclProSettings(listView1.clProSettings);
  designerPanel1 = anaForm.AddNewProListViewDesignerPanel(listView1,'designerPanel1');
  listView1.SetDesignerPanel(designerPanel1);

  anaForm.Run;
}
```

DesignerPanel'e belirli miktarda öğe ekleyebiliriz. Eklenecek bu elemanların bağlama parametreleri aşağıdaki gibidir:

- clCaption
- clText
- clText1
- clText2
- clText3
- clText4
- clText5
- clText6
- clImage1
- clImage2
- clRecord_GUID

Bu parametreleri kullanabilmek için bir bileşen tanımlanmaktadır. Örneğin bileşen adı *testLbl* olan bir label ve bu bileşen *ProListViewDesignerPanel* içerisinde bulunmasını istiyorsak *AddPanelObject* fonksiyonuna atama yapılmalıdır. Bu fonksiyonun ilk parametresine eklenecek bileşen *testLbl*, ikinci parametre ise yukarıdaki alanlardan hangisine eklemek isteniyorsa yazılmalıdır. Unutmayalım ki her alan bir kez kullanılmaktadır.

```
designerPanel1. AddPanelObject (testLbl, clText );
```

Bileşen içerisinde nesnelerin eklenebilmesi için veri kaynağındaki alan ismi, değişken tanımlarken de kullanılmalıdır. Yani veri tabanından gelen alan adı ile projenizde yer alan değişken adı aynı olmalıdır.

Örneğin; ‘designerPanel1’ içerisine bir tane bileşen oluşturuldu. Bu bileşene veri tabanından alan adı *urunAdi* olan veri gelmektedir. Uygulamada TclLabel nesnesi oluşturup veri tabanından gelen alan adı ile aynı olmalıdır. Yani değişkenin ismi urunAdi olmalıdır. Aşağıda bu kısmı anlatan kod parçası verilmiştir.

```
Var
  anaForm:TclForm;
  listView1 : TClProListView;
  designerPanel1 : TClProListViewDesignerPanel;
  urunAdi: TClProLabel;

void listViewEkle;
{
  listView1 = anaForm.AddNewProListView(anaForm,'listView1');
  listView1.Align = AlClient;
  listView1.clProSettings.ViewType = lvIcon;
  listView1.clProSettings.BorderColor = clAlphaColor.clHexToColor('#0850c4');
  listView1.clProSettings.BorderWidth = 3;
  listView1.SetclProSettings(listView1.clProSettings);
}

void designerPanelEkle;
{
  designerPanel1 =
anaForm.AddNewProListViewDesignerPanel(listView1,'designerPanel1');
  listView1.SetDesignerPanel(designerPanel1);
}

//designerPanel içerisine urunAdi değişkeni ekleniyor.
void urunAdiEkle;
 {
    urunAdi = anaForm.AddNewProLabel(designerPanel1, 'urunAdi','test');
    designerPanel1.AddPanelObject(urunAdi,clCaption);
    urunAdi.Properties.AutoSize = True;
 }

//veri tabanı eklenmesi
void ListViewVeriEkle;
Var
  foodListQuery : TClSqlQuery;
{
  foodListQuery = TClSqlQuery.Create(nil);
  Clomosy.DBSQLServerConnect('SQL Server', 'sunucu_adi', 'kullanici_adi',
'kullanici_sifre','veritabani_adi', 1433);
```

```
  try
    foodListQuery.Connection = Clomosy.DBSQLServerConnection;
    foodListQuery.SQL.Text = 'SELECT urunAdi from Urunler';
    foodListQuery.Open;
    if (foodListQuery.Found)
    {
      listView1.clLoadProListViewDataFromDataset(foodListQuery);
    }
  finally
    foodListQuery.Close;
    foodListQuery.Free;
  }
 }

{
  anaForm = TclForm.Create(Self);
  listViewEkle;
  designerPanelEkle;
  urunAdiEkle;
  ListViewVeriEkle;

  anaForm.Run;
}
```

10.4. Veri İçeren Bileşenler

Bunlara örnek olarak; 'TclEdit, TclMemo, TclSearchBox, TclProDateEdit' verilmektedir. Kullanıcı tarafından bu verilere doğrudan veri girişi yapılabilirken, 'TclLabel ve TclProLabel' bileşenlerine yönelik veri girişini tasarım aşamasında 'text' özelliğine atama yaparak ya da çalışma anında kod kullanarak gerçekleştirilmektedir.

10.4.1. TclEdit

Tek satırlık metinsel veri giriş işlemlerinde kullanılan bileşendir. Bir TclEdit nesnesini oluşturmak için *AddNewEdit* fonksiyonu kullanılmaktadır. Bu işlev üç parametre almaktadır. İlk parametre tanımlanacak olan nesnenin hangi bileşen içerisinde konumlanacağını belirlemektedir. İkinci parametre, tanımlanan nesnenin ismini temsil etmektedir. Üçüncü parametre ise nesnenin bilgi veren metnini vermektir.

Örnekte edit üzerinden isim alabileceğimiz bir durum sağlayalım. *testEdit* bileşeninden isim alınacağı için üçüncü parametre olarak ekranda görünen içerik kısmı *Adınızı giriniz…* olarak yazılmıştır.

```
var
  anaForm:TclForm;
  testEdit : TclEdit;
```

```
{
  anaForm = TclForm.Create(Self);

  testEdit= anaForm.AddNewEdit(anaForm,'testEdit','Adınızı giriniz...');

  anaForm.Run;
}
```

10.4.2. TClProEdit

Standart metin girişinin (TclEdit) profesyonel halidir. Standart yapıdan farkı kenarlık ve arka plan rengi verme gibi farklı özellikler bulunmaktadır. Bileşeni oluşturmak için *AddNewProEdit* fonksiyonu kullanılmaktadır.

```
Var
  anaForm:TclForm;
  testEdit: TClProEdit;

{
  anaForm = TclForm.Create(Self);
  testEdit = anaForm.AddNewProEdit(anaForm,'testEdit','Adınızı giriniz...');
  anaForm.Run;
}
```

10.4.3. TclMemo

Birden fazla satırlı metinsel veri giriş işlemlerinde görev almaktadır. Bir TclMemo nesnesini oluşturmak için *AddNewMemo* fonksiyonu kullanılmaktadır. Bu işlev üç parametre almaktadır. İlk parametre tanımlanacak olan nesnenin hangi bileşen içerisinde konumlanacağını, ikinci parametre ismini, üçüncü parametre ise bilgi veren metnini ifade etmektedir.

```
var
  anaForm : TclForm;
  testMemo : TclMemo;

{
  anaForm = TclForm.Create(Self);

  testMemo = anaForm.AddNewMemo(anaForm,'testMemo','Bir not ekleyiniz…');
  testMemo.Align = alCenter;
  testMemo.Height = 50;
  testMemo.Width = 200;

  anaForm.Run;
}
```

TclMemo kontrolünün içindeki satırlar alınmak veya indirilmek istendiğinde *Lines* özelliği kullanılmaktadır. Aşağıda nesnenin bazı özelliklerden bahsedilmektedir.

Örnek olarak ‘testMem’o üzerinde uygulama çalıştırılırken *Add* fonksiyonu ile T0, T1, T2, T3 yazıldı diyelim. Onun üzerinden özellikleri deneyelim.

Özellik	Amacı
testMemo.Lines.Add('T0');	Memo da ekleme işlemi yapılır.
testMemo.Lines.Count	Memodaki satır sayısını verir.
testMemo.Lines.Delete(1);	Memoda silme işlemi yapar. Örnekte index 1 olan silindi. Sonuç: T0, T2, T3 kalır.
testMemo.Lines.Exchange(0,2);	İki satırın konumlarını belirtilen dizinlerle değiştirir. Örnekte index 0 ile 2 yer değiştirir. Sonuç: T2, T1, T0, T3 olur.
testMemo.Lines.Move(1,3);	Satır konumu değiştirilir. İlk parametre yer değiştirilecek satır, ikinci parametre ise değişecek konum. Örnekte İndex 1 de olan veri index 3’e taşınır. Sonuç: T0, T2, T3, T1
testMemo.Lines.Clear;	Memo üzerindeki bütün verileri siler.
testMemo.Lines.Append('T4');	Memo sonuna yeni bir satır eklenir. Sonuç: T0, T1, T2, T3, T4
testMemo.Lines.Insert(2,'T5');	İstenilen satıra yeni veri eklenir. Örnekte index 2 olan satıra T5 verisini ekle. Sonuç: T0, T1, T5, T2, T3

10.4.4. TclSearchBox

Kullanıcıya bir metin listesi içinde arama yapma seçeneği sunmaktadır. Kısaca kullanıcının bir listede belirli bir terimi aramasına ve bu terimi içeren öğeleri filtrelemesine olanak tanımaktadır.

Kullanım kolaylığı sağlamak ve arama işlemlerinin verimliliğini artırmak için tasarlanan *TclSearchBox*'ın kullanıcı dostu olması amaçlanmıştır. Kullanıcı *TclSearchBox*'a metin girdikten sonra, girilen metinle eşleşen öğelerin listesini görüntülemek için bileşen bir *TclListView* bileşeniyle entegre edilmektedir.

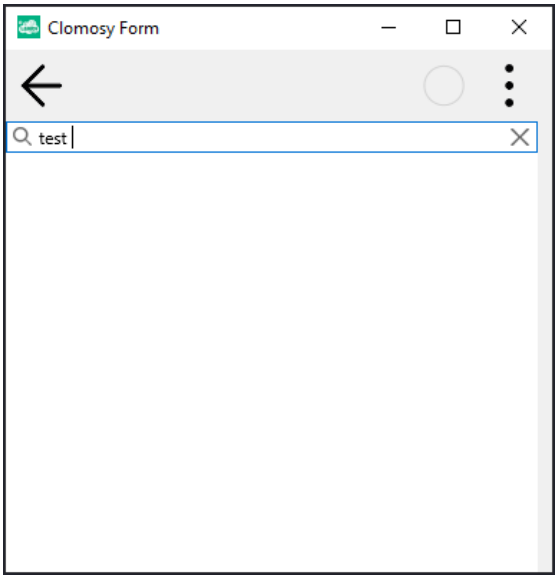
TclSearchBox değişkeni tanımlaması aşağıdaki gibi yapılmaktadır.

```
var
  testSearchBox : TclSearchBox;
```

ListView bileşeni içindeki bir arama kutusuna *GetSearchBox* özelliği kullanılarak erişilmektedir. Bu yöntem sayesinde TclListView arama çubuğu artık *testSearchBox* nesnesine bağlanmış olur.

Aşağıdaki örnekte TclListView nesnesi oluşturulur. TclSearchBox, nesneye bağlanır.

```
Var
  anaForm:TclForm;
  testListview : TClListView;
  testSearchBox : TclSearchBox;
{
  anaForm = TclForm.Create(Self);

  testListview = anaForm.AddNewListView(anaForm,'testListview');
  testListview.Align = alClient;
  testSearchBox = testListview.GetSearchBox;

  anaForm.Run;
}
```



Resim 51

Yukarıda görüldüğü gibi düz bir arama çubuğu sağlamış olduk. Burada ne yazılırsa TclListView nesnesi içerisindeki verilerde arama yapmaya çalışacaktır.

10.4.5. TClProDateEdit

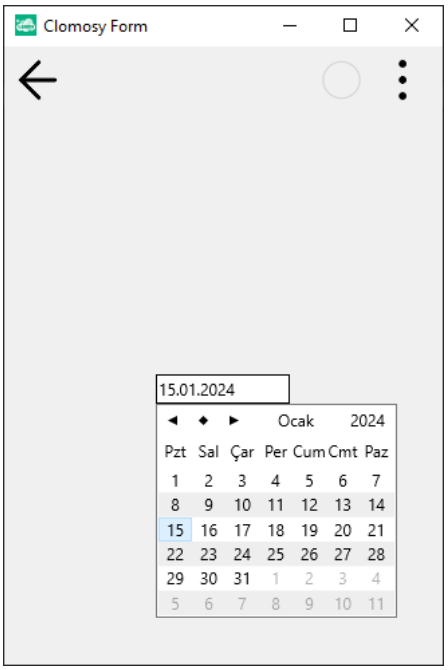
Kullanıcının bir tarih seçmesine izin veren bir giriş kutusu ve tarih seçimi için bir takvim penceresi içermektedir. Kullanıcılar, takvim penceresinden tarih seçebilir veya doğrudan edit kutusuna tarih girebilirler.

Bu bileşen, genellikle tarihle ilgili veri girişi gerektiren uygulamalarda kullanılmaktadır. Örneğin, bir rezervasyon uygulamasında veya bir iş takviminde tarih seçimi için kullanılmaktadır.

AddNewProDateEdit fonksiyonu ile oluşturulmaktadır. Bu işlev iki parametre almaktadır. İlk parametre tanımlanacak olan nesnenin hangi bileşen içerisinde konumlanacağını belirlemektedir. İkinci parametre ise nesnenin ismini temsil etmektedir.

```
Var
  anaForm:TclForm;
  testDate: TClProDateEdit;

{
  anaForm = TclForm.Create(Self);

  testDate = anaForm.AddNewProDateEdit(anaForm,'testDate');

  anaForm.Run;
}
```

Kodun çıktısı aşağıdaki gibidir.



Resim 52

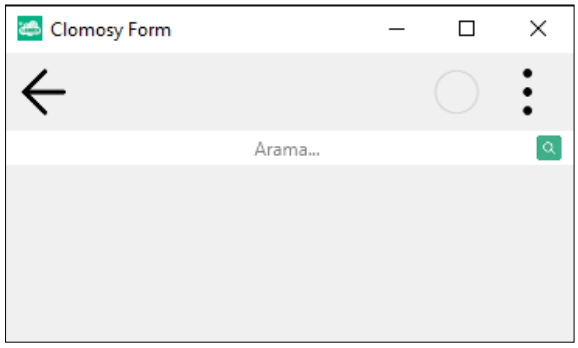
10.4.6. TClProSearchEdit

Kullanıcının bir metin listesi içinde arama yapmasını sağlamaktadır. *TclSearchBox* bileşeniyle aynı özelliklere sahiptir. Bu bileşenden ayrılan özellikleri de bulunmaktadır. Görsellik olarak farklı tasarımlar yapılabilmesinin dışında barkod okuma özelliği de vardır. Bu bileşen *TclProListView* için kullanılmaktadır. *TclProListView* nesnesi oluşturulduğunda arama çubuğu gelmemektedir. Arama çubuğu için bu bileşen kullanılmaktadır.

AddNewProSearchEdit fonksiyonu ile arama çubuğu oluşturulmaktadır. Bu işlev üç parametre almaktadır. İlk parametre tanımlanacak olan arama çubuğunun hangi bileşen içerisinde konumlanacağını belirlemektedir. İkinci parametre, tanımlanan arama çubuğunun ismini temsil etmektedir. Üçüncü parametre ise arama çubuğuna bilgi veren metni vermektir.

Arama çubuğuna hedef TclProListView belirlemek için *TargetListView* özelliğine atama yapmak gerekmektedir.

```
Var
  anaForm:TclForm;
  testListview : TClProListView;
  aramaEdt : TClProSearchEdit;

void aramaEdtGetir;
{
  aramaEdt = anaForm.AddNewProSearchEdit(anaForm,'aramaEdt','Arama...');
  aramaEdt.Align = alTop;
}

void listViewOlustur;
{
  testListview = anaForm.AddNewProListView(anaForm,'testListview');
  testListview.Align = AlClient;
  testListview.ListType = 'horizontal';
  aramaEdt.TargetListView = testListview;
}

{
  anaForm = TclForm.Create(Self);
  aramaEdtGetir;
  listViewOlustur;
  anaForm.Run;
}
```



Resim 53

Arama çubuğuna barkod okuma özelliği getirebilmek için *clBarcodeReader* özelliğine True değeri atanmaktadır. Aşağıda değişim gösterilmiştir.

```
aramaEdt.clBarcodeReader = True;
```

Resim 54

10.4.7. TclLabel

Genellikle bir form veya başka bir konteyner içinde bulunan metni göstermektedir. Bu metin, genellikle kullanıcıya bir kontrolün amacını veya üzerindeki diğer kontrol öğeleri hakkında bilgi vermek için kullanılmaktadır.

AddNewLabel fonksiyonu ile bir TclLabel nesnesi oluşturulmaktadır. Bu işlev üç parametre almaktadır. İlk parametre, tanımlanacak olan nesnenin hangi bileşen içerisinde konumlandırılacağını belirler. İkinci parametre, tanımlanan nesnenin ismini temsil eder. Üçüncü parametre ise nesneye içerik adlandırması yapılır.

```
var
  anaForm:TclForm;
  testLabel : TclLabel;

{
  anaForm = TclForm.Create(Self);

  testLabel = anaForm.AddNewLabel(anaForm,'testLabel','Label Başlık');

  anaForm.Run;
}
```

10.4.8. TclProLabel

TclLabel gibi form veya başka bir konteyner içinde bulunan metni göstermek için kullanılmaktadır. TclLabel özelliklerinin hepsini kapsar. Bunun dışında nesnenin arka plan rengi, kalın metin, kenarlık rengi, otomatik boyut vb. özel kullanımları da bulunmaktadır.

```
var
  anaForm:TclForm;
  testLabel : TclProLabel;

{
  anaForm = TclForm.Create(Self);
```

```
  testLabel = anaForm.AddNewProLabel(anaForm,'testLabel','ProLabel Başlık');

  anaForm.Run;
}
```

10.5. Form ve Panel Tipi Bileşenler

Bir uygulama genellikle en az bir form yapısından oluşur ve doğal olarak tek bir kullanıcı arayüzüne sahiptir. İhtiyaçlara bağlı olarak, uygulama arayüzü çeşitli alt ekranlar kullanılarak oluşturulabilir. Alt ekranların tasarımı, ihtiyaçlar ve modül sayıları göz önüne alınarak dikkatlice yapılmalıdır. Tasarım sürecinin önemli bir kısmı kâğıt üzerinde tamamlanmalı ve kodlamaya başlamadan önce bitirilmelidir. İş süreçlerine göre yapılan tasarımlar, karmaşıklıklara neden olabileceğinden özenle ele alınmalıdır. Ana ekran genellikle bir formu içerir, diğer alt ekranlar ise Layout, Panel gibi bileşenlerden oluşabilir.

10.5.1. TclPanel

TclPanel bileşeni; kontrol öğelerini gruplama, düzenleme ve sınırlama işlevlerini yerine getirmektedir. Aynı zamanda bir form üzerinde, belirli bir işlevselliğe sahip bir bölge oluşturmaktadır. Genelde Windows için geliştirilen uygulamalarda kullanılmaktadır.

AddNewPanel fonksiyonu ile panel oluşturulmaktadır. İki parametre alan bu işlevde ilk parametre; panelin hangi bileşen içinde konumlanacağını belirlemektedir. İkinci parametre, tanımlanan panel ismini temsil etmektedir.

```
Var
  anaForm:TclForm;
  testPanel : TclPanel;

{
  anaForm = TclForm.Create(self);

  testPanel = anaForm.AddNewPanel(anaForm,'testPanel');

  anaForm.Run;
}
```

10.5.2. TclProPanel

TclPanel gibi kontrol öğelerini gruplamak, düzenlemek ve sınırlamak için kullanılmaktadır. *TclPanel* özelliklerinin hepsini kapsar. TclProPanel nesnenin arka plan rengi, kenarlık rengi vb. özel kullanımları da bulunmaktadır.

```
Var
  anaForm:TclForm;
  testPanel : TclProPanel;

{
  anaForm = TclForm.Create(self);
    testPanel = anaForm.AddNewProPanel(anaForm,'testPanel');

  anaForm.Run;
}
```

10.5.3. TclRectangle

TclRectangle bileşeni, bir dikdörtgeni temsil eder ve genellikle arayüz tasarımlarında arka plan veya bölge olarak kullanılır. Ayrıca, görünümü ve özellikleri stil özelliklerini kullanarak özelleştirmek mümkündür.

Çok platformlu uygulamalar oluşturmak için kullanılan bir çerçevedir ve farklı platformlarda (Windows, iOS, Android vb.) çalışan uygulamalar geliştirmek için kullanılır.

Bir nesne oluşturmak için *AddNewRectangle* fonksiyonu kullanılır. Bu işlev iki parametre almaktadır. İlk parametre tanımlanacak olan nesnenin hangi bileşen içerisinde konumlandırılacağını belirler. İkinci parametre nesnenin ismini temsil eder.

```
var
  AnaForm :TclGameForm;
  Rectangle1 : TclRectangle;

{
   AnaForm = TCLGameForm.Create(Self);
   Rectangle1= MyForm.AddNewRectangle(AnaForm, 'Rectangle1');
   AnaForm.Run;
}
```

TclRectangle nesnesinin içini doldurmak için "*Fill.Kind"* özelliği kullanılır. Bu özellik ile nesnenin iç dolgu şekli belirlenir. Aşağıdaki özellikler kullanılabilir.

Özellik	Açıklama
fbkNone	İçini doldurmaz.
fbkSolid	Tek bir renk ile içini doldurur.
fbkGradient	Bir renk gradyanı ile içini doldurur.
fbkBitmap	Bir bit işlem ile içini doldurur.
fbkResource	Kaynak dosya ile doldurur (genellikle resim dosyaları, metin dosyaları, veri tabanı bağlantıları veya benzeri şeyleri içerebilir).

Kullanımı:

```
Rectangle1.Fill.Kind = fbkSolid;
```

TclRectangle bileşeninin içini doldurmak için kullanılan bir bit eşlemi (bitmap) belirlerken, bu bitmap'in nasıl örtüleceğini kontrol etmek için *Fill.Bitmap.WrapMode* özelliği kullanılır. Bit eşleminin belirli bir alana nasıl yerleştirileceğini ve nasıl örtüleceğini belirler. Aşağıdaki özellikler kullanılabilir.

Özellik	Açıklama
fbwmTile	Bit eşlem, içi doldurulacak alana tamamen sığacak şekilde örtülür.
fbwmTileOriginal	Bit eşlem, orijinal boyutlarına uygun olarak örtülür ve içi doldurulacak alanın sınırlarını aşarsa, tekrar eden desen oluşturarak örtülür.
fbwmTileStretch	Bitmap'in özellikle boyutlarını örtülecek alana sığdırmak üzere gerilir (stretching) ve örtülen alana tamamen yerleştirilir.

Kullanımı:

```
Rectangle1.Fill.Bitmap.WrapMode = fbwmTileOriginal;
```

Nesnenin çerçeve kalınlığı belirlemek için **Stroke.Thickness** özelliği kullanılır. 0 değeri çerçeveyi kapatır.

```
Rectangle1.Stroke.Thickness = 3;
```

Nesnenin her bir kenarı için genişlik ayarlamalarını özelleştirmek için *Sides* özelliği kullanır.

```
Nesne1.Sides = SetOf([sdLeft, sdTop, sdRight, sdBottom]);
```

"SetOf" diyerek dört kenarın boyutları belirlenir. Yukarıda görüldüğü gibi sırayla sol, üst, sağ, alt kenarlar için ilgili parametreler verilir. Çerçevesiz olması istenilen kenarlar için ilgili parametre silinir.

Aşağıda özelliklerine ait örnek kullanım mevcuttur.

```
var
  anaForm:TclGameForm;
  Rectangle1: TclRectangle;

{
  anaForm = TclGameForm.Create(Self);

  Rectangle1 = anaForm.AddNewRectangle(anaForm, 'Rectangle1');
  Rectangle1.Fill.Kind = fbkBitmap;
  Rectangle1.Fill.Bitmap.WrapMode = fbwmTile;
  Rectangle1.Width = 200;
  Rectangle1.Height = 200;
  Rectangle1.Stroke.Thickness = 3;
```

```
  Rectangle1.Sides = SetOf([sdLeft, sdTop, sdBottom]);

  anaForm.Run;
}
```

10.5.4. TclCircle

TclCircle, bir daire veya elips şeklinde bir grafik öğesini temsil eder. Uygulamalarında kullanıcı arayüzü tasarımında çeşitli görsel efektler ve özellikler eklemek için kullanılabilir.

Bir *TclCircle* nesnesi oluşturmak için *AddNewCircle* fonksiyonu kullanılır. Bu işlev iki parametre almaktadır. İlk parametre tanımlanacak olan nesnenin hangi bileşen içerisinde konumlandırılacağını belirler. İkinci parametre nesnenin ismini temsil eder.

```
var
AnaForm :TclGameForm;
Circle1 : TclRectangle;
{
  AnaForm = TCLGameForm.Create(Self);
  Circle1 = AnaForm.AddNewCircle(AnaForm, 'Circle1');
  AnaForm.Run;
}
```

TclCircle nesnesinin içini doldurmak için "*Fill.Kind"* özelliği kullanılır. Bu özellik ile nesnenin iç dolgu şekli belirlenir. Aşağıdaki özellikler kullanılabilir.

Özellik	Açıklama
fbkNone	İçini doldurmaz.
fbkSolid	Tek bir renk ile içini doldurur.
fbkGradient	Bir renk gradyanı ile içini doldurur.
fbkBitmap	Bir bit işlem ile içini doldurur.
fbkResource	Kaynak dosya ile doldurur (genellikle resim dosyaları, metin dosyaları, veri tabanı bağlantıları veya benzeri şeyleri içerebilir).

Kullanımı:

```
Circle1.Fill.Kind = fbkSolid;
```

TclCircle bileşeninin içini doldurmak için kullanılan bir bit eşlemi (bitmap) belirlerken, bu bitmap'in nasıl örtüleceğini kontrol etmek için "*Fill.Bitmap.WrapMode"* özelliği kullanılır. Bit eşleminin belirli bir alana nasıl yerleştirileceğini ve nasıl örtüleceğini belirler. Aşağıdaki özellikler kullanılabilir.

Özellik	Açıklama
fbwmTile	Bit eşlem, içi doldurulacak alana tamamen sığacak şekilde örtülür.
fbwmTileOriginal	Bit eşlem, orijinal boyutlarına uygun olarak örtülür ve içi doldurulacak alanın sınırlarını aşarsa, tekrar eden desen oluşturarak örtülür.
fbwmTileStretch	Bitmap'in özellikle boyutlarını örtülecek alana sığdırmak üzere gerilir (stretching) ve örtülen alana tamamen yerleştirilir.

Kullanımı:

```
Circle1.Fill.Bitmap.WrapMode = fbwmTileOriginal;
```

Nesnenin çerçeve kalınlığı belirlemek için **Stroke.Thickness** özelliği kullanılır. 0 değeri çerçeveyi kapatır.

```
Circle1.Stroke.Thickness = 3;
```

Aşağıda örnek kullanım mevcuttur.

```
var
  AnaForm:TclGameForm;
  Circle1: TclCircle;

{
  AnaForm = TCLGameForm.Create(Self);

  Circle1 = AnaForm.AddNewCircle(AnaForm, 'Circle1');
  Circle1.Fill.Kind = fbkBitmap;
  Circle1.Fill.Bitmap.WrapMode = fbwmTileOriginal;
  Circle1.Width = 200;
  Circle1.Height = 200;
  Circle1.Stroke.Thickness = 3;

  AnaForm.Run;
}
```

10.5.5. TclLayout

TclLayout, kontrol öğelerini düzenlemek, gruplamak ve belirli bir düzeni korumak için kullanılmaktadır. Aynı zamanda farklı ekran boyutlarına ve cihazlara uyum sağlamak amacını da taşır. *TclPanel* ile benzerdir, ancak genellikle *TclPanel* Windows uygulamalarında tercih edilirken, *TclLayout* mobil uygulamalarda kullanılan bir bileşendir.

AddNewLoyut fonksiyonu ile nesne oluşturulmaktadır. İki parametre alan bu işlevin ilk parametresi; tanımlanacak nesnenin hangi bileşen içinde konumlanacağını, ikinci parametre ise tanımlanan nesnenin ismini temsil etmektedir.

```
Var
  anaForm:TclForm;
  testLayout : TclLayout;

{
  anaForm = TclForm.Create(self);

  testLayout = anaForm.AddNewLayout(anaForm,'testLayout');

  anaForm.Run;
}
```

10.5.6. TclHorzScrollBox

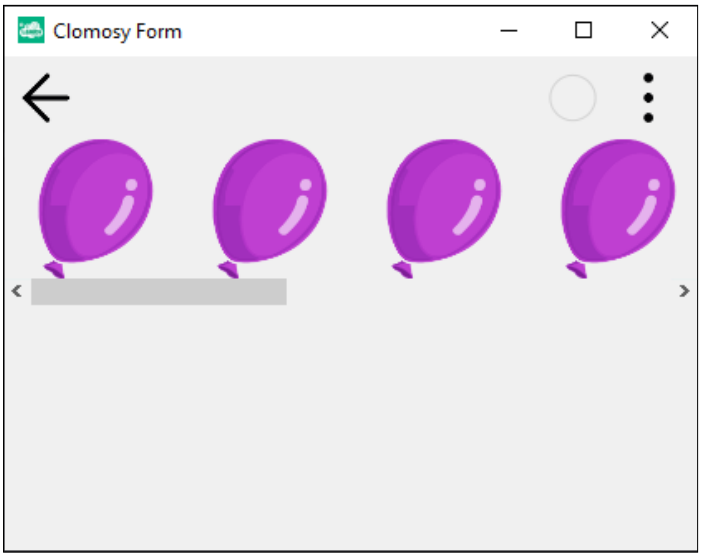
Clomosy bileşenleri arasında yatay kaydırma kutusu (horizontal scroll box) bulunmaktadır. Bu bileşen, içerdiği kontrol öğelerini düzenlemek ve bir yatay kaydırma çubuğu ile içerik arasında gezinme sağlamak için kullanılmaktadır.

Aşağıdaki örnekte bir *TclHorzScrollBox* nesnesi oluşturulup içerisine 10 adet görsel eklenmiştir.

```
var
  anaForm:TclForm;
  horzScrollBox : TClHorzScrollBox;
  testImg : TClProImage;
  i : Integer;

{
  anaForm = TclForm.Create(Self);

  horzScrollBox = anaForm.AddNewHorzScrollBox(anaForm,'horzScrollBox');
  horzScrollBox.Align = alMostTop;
  horzScrollBox.Height = 100;
  horzScrollBox.ShowScrollBars = True;

  for (i = 1 to 10)
  {
```

```
    testImg = anaForm.AddNewProImage(horzScrollBox,'testImg'+IntToStr(i));
    testImg.Align = alLeft;
    testImg.Margins.Left = 5;
    testImg.Width = 100;
    anaForm.SetImage(testImg,'https://clomosy.com/demos/balloon.png');
  }

  anaForm.Run;
}
```

Yukarıdaki örneğinin çıktısı aşağıda gösterilmektedir.



Resim 55

10.5.7. TclVertScrollBox

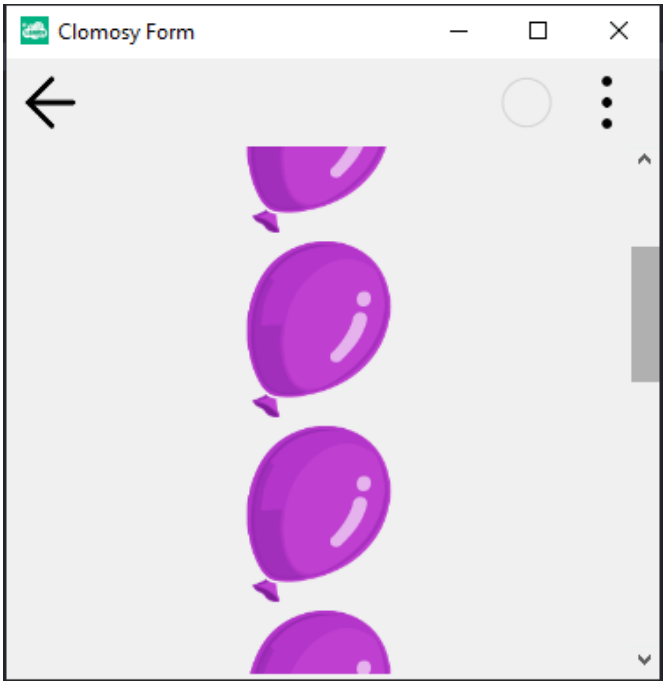
Clomosy bileşenleri arasında dikey kaydırma kutusu (vertical scroll box) bileşeni bulunmaktadır. Bu bileşen, içerdiği kontrol öğelerini düzenleme, bir dikey kaydırma çubuğu ile içerik arasında gezinme sağlamak amacıyla kullanılmaktadır.

Aşağıdaki örnek, *TclHorzScrollBox* örneğinin *TclVertScrollBox* ile kullanımı aşağıda yer almaktadır.

```
var
  anaForm:TclForm;
  vertScrollBox : TclVertScrollBox;
  testImg : TClProImage;
  i : Integer;

{
  anaForm = TclForm.Create(Self);
  vertScrollBox = anaForm.AddNewVertScrollBox(anaForm,'vertScrollBox');
```

```
  vertScrollBox.Align = alMostTop;
  vertScrollBox.Height = 300;
  vertScrollBox.ShowScrollBars = True;

  for (i = 1 to 10)
  {
    testImg = anaForm.AddNewProImage(vertScrollBox,'testImg'+IntToStr(i));
    testImg.Align = alTop;
    testImg.Margins.Top = 5;
    testImg.Height = 100;
    anaForm.SetImage(testImg,'https://clomosy.com/demos/balloon.png');
  }
  anaForm.Run;
}
```

Aşağıda çıktısı gösterilmektedir.



Resim 56

10.5.8. TclPageControl

TclPageControl, çok sayfalı bir iletişim kutusu oluşturmak için kullanılan bir sayfa kümesidir. Kullanıcıların birden fazla sayfa arasında geçiş yapmasına olanak tanır.

Örneğin, çok sayfalı bir iletişim kutusu veya sekmeli bir not defteri oluşturmak için kullanılır. *TclPageControl*, birden fazla üst üste binen sayfa görüntüler. Kullanıcı, denetimin en üstünde görünen sayfa sekmesine tıklayarak bir sayfa seçer.

AddNewPageControl fonksiyonu ile sayfa oluşturulmaktadır. İki parametre alan bu işlevin ilk parametresi; tanımlanacak bileşenin ebeveynini, ikinci parametre ise tanımlanan bileşenin ismini temsil etmektedir.

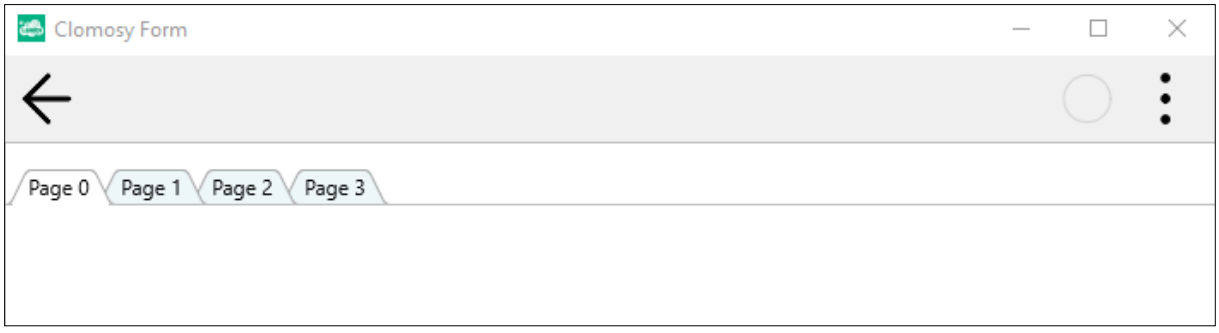
AddNewPageControl(TclComponent,xName): TclPageControl

Tasarım zamanında *TclPageControl* nesnesine yeni bir sayfa eklemek için *TClPageControlContainer* ile sayfalar oluşturulur.

Aşağıda 3 sayfanın oluşturulmasını gösteren basit bir oluşturma süreci gösterilmektedir.

NOT: *TclPageControl* nesnesi oluşturulduğunda, otomatik olarak varsayılan bir sayfa atar. *TClPageControlContainer* nesnesiyle oluşturulan kadar sayfa ekler. Aşağıdaki örnekte 3 sayfa oluşturduk ancak varsayılan bir sayfa olduğu için 4 sayfa olarak görünecektir.

```
var
  Form1:TCLForm;
  clPageControl:TCLPageControl;
  clPageControlContainer:TClPageControlContainer;
 {
  Form1 = TclForm.Create(Self);

  clPageControl = Form1.AddNewPageControl(Form1,'clPageControl');
  clPageControl.Align = alClient;

  clPageControlContainer = clPageControl.AddContainer;
  clPageControlContainer = clPageControl.AddContainer;
  clPageControlContainer = clPageControl.AddContainer;

  Form1.Run;
 }
```



Ayrıntılı bilgiye doküman sitesinden (https://www.docs.clomosy.com) erişim sağlanabilir.

10.6. Donanım Etkileşimli Bileşenler

Analog cihaz ve makinelerin birbirleriyle veya canlılarla iletişim kurması imkansızdır. Etkili bir iletişim kanalı veya protokolü kullanarak, cihazlar arasında hem iletişim sağlanabilir hem de bu iletiler anlamlandırılarak bizim anlayabileceğimiz bir formata dönüştürülebilir. Buna en iyi örnek IoT (Internet of Things)'dir. Bu teknoloji, cihazların insanlara bilgi iletmesini ve insanların cihazları uzaktan kontrol edebilmelerini sağlamaktadır.

10.6.1. TclOpenAIEngine

Clomosy altyapısı içerisinde OpenAI motoru kullanılarak diyalog ve konuşmaya dayalı etkileşimlere odaklanan özel bir yapay zekâ chatbotu geliştirildi. *TclOpenAIEngine* sohbet motoru aracılığıyla kullanılmaktadır.

TclOpenAIEngine sınıfına ait bir değişken tanımlanır ve nesneyi oluşturmak için *Create* işlemi gerçekleştirilir. Yapay zekâ motorunu aktifleştirmek için *SetToken* özelliğine bir token aktarılmalı ve ayrıca bileşenin *ParentForm* özelliği tanımlanmalıdır. Yapay zekadan dönen mesajları alabilmek için ise *OnNewMessageEvent* olayına bir tetikleme prosedürü atanmalıdır. Bu tetikleme olayı, yapay zekadan bir mesaj geldiğini belirtir. Yapay zekadan gelen mesajı almak için ise *NewMessageContent* özelliği kullanılır.

i	*SetToken* yöntemi bir nesne için tanımlanmalıdır ve bu yönteme parametre olarak sağlanan değer, **OpenAI** hizmetiyle kimlik doğrulama için kullanılan bir belirteci temsil eder. Bu sebepten ötürü Token alabilmek için https://openai.com sitesinden giriş yaparak ücretsiz token alınabilmektedir.

Bu özelliklerin kullanımına bir örnek üzerinde bakalım. Örnekte, en üst kısımda yapay zekaya sorulacak mesaj yazılmalıdır. Ardından *Gönder* butonuna tıklandığı anda alt kısımda bulunan memoya mesaj yazılır. Yapay zekadan mesajınıza karşılık gelir. Bunları görebilmek için OpenAI sitesinden token almanız gerekmektedir.

```
var
 MyForm:TclForm;
 BtnSend:TclProButton;
 MemMsg:TclMemo;
 chatSectionMemo:TclMemo;
 MyOpenAIEngine:TclOpenAIEngine;
 bigPanel,middlePanel:TclProPanel;
 bigLyt:TClLayout;
 MyMQTT : TclMQTT;

 void BtnSendClick;
 {
    if (MemMsg.Text == '')
    {
       ShowMessage('Write a message!');
```

```
    }
    else
    {
       MyOpenAIEngine.SendAIMessage(MemMsg.Text);
       MemMsg.Text = '';
    }
 }
void OnNewMessageEvent;
 {

    if (not MyMQTT.ReceivedAlright)
    {
      chatSectionMemo.Lines.Add('');
      MyMQTT.Send(MyOpenAIEngine.NewMessageContent);
  chatSectionMemo.Lines.Add(MyOpenAIEngine.NewMessageContent);

chatSectionMemo.ScrollTo(0,chatSectionMemo.Lines.Count*chatSectionMemo.Lines.Cou
nt,True);
    }
 }

 void MyMQTTPublishReceived;
 {
      If (MyMQTT.ReceivedAlright)
      {
        chatSectionMemo.Lines.Add('');
        chatSectionMemo.Lines.Add('                ' + MyMQTT.ReceivedMessage);

chatSectionMemo.ScrollTo(0,chatSectionMemo.Lines.Count*chatSectionMemo.Lines.Cou
nt,True);
      }
 }

{
 MyForm = TclForm.Create(Self);

 bigLyt = MyForm.AddNewLayout(MyForm,'bigLyt');
 bigLyt.Align=alContents;
 bigLyt.Margins.Left=10;
 bigLyt.Margins.Right=10;
 bigLyt.Margins.Top=10;
 bigLyt.Margins.Bottom=30;

 MyMQTT = MyForm.AddNewMQTTConnection(MyForm,'MyMQTT');

MyForm.AddNewEvent(MyMQTT,tbeOnMQTTPublishReceived,'MyMQTTPublishRecei
ved');
```

```
MyMQTT.Channel = 'ChatAI';
 MyMQTT.Connect;

 middlePanel=MyForm.AddNewProPanel(bigLyt,'middlePanel');
 middlePanel.Align = AlTop;
 middlePanel.Width = 300;
 middlePanel.Height = 50;
 middlePanel.Margins.Right = 10;
 middlePanel.Margins.Top = 30;
 middlePanel.Margins.Left = 10;
 middlePanel.clProSettings.IsRound = True;
 middlePanel.clProSettings.RoundHeight = 10;
 middlePanel.clProSettings.RoundWidth = 10;
 middlePanel.clProSettings.BorderColor = clAlphaColor.clHexToColor('#008000');
 middlePanel.clProSettings.BorderWidth = 3;
 middlePanel.SetclProSettings(middlePanel.clProSettings);

 MemMsg= MyForm.AddNewMemo(middlePanel,'MemMsg','');
 MemMsg.Align = alTop;
 MemMsg.Margins.Right=10;
 MemMsg.Margins.Left=10;
 MemMsg.Margins.Top=10;
 MemMsg.Margins.Bottom= 10;

 bigPanel=MyForm.AddNewProPanel(bigLyt,'bigPanel');
 bigPanel.Align = AlClient;
 bigPanel.Margins.Right = 10;
 bigPanel.Margins.Left = 10;
 bigPanel.clProSettings.IsRound = True;
 bigPanel.clProSettings.RoundHeight = 10;
 bigPanel.clProSettings.RoundWidth = 10;
 bigPanel.clProSettings.BorderColor = clAlphaColor.clHexToColor('#008000');
 bigPanel.clProSettings.BorderWidth = 3;
 bigPanel.SetclProSettings(bigPanel.clProSettings);

 chatSectionMemo= MyForm.AddNewMemo(bigPanel,'chatSectionMemo','');
 chatSectionMemo.Align = alClient;
 chatSectionMemo.ReadOnly = True;
 chatSectionMemo.Margins.Top= 10;
 chatSectionMemo.Margins.Left= 10;
 chatSectionMemo.Margins.Right =10;
 chatSectionMemo.Margins.Bottom =10;
 chatSectionMemo.TextSettings.Font.Size=26;
 chatSectionMemo.TextSettings.WordWrap = True;
 chatSectionMemo.EnabledScroll = True;
```

```
 MyOpenAIEngine=TclOpenAIEngine.Create(Self);
 MyOpenAIEngine.ParentForm = MyForm;
 MyOpenAIEngine.SetToken('AIToken');
 MyOpenAIEngine.OnNewMessageEvent = 'OnNewMessageEvent';

 BtnSend = MyForm.AddNewProButton(bigLyt,'BtnSend','SEND');
 BtnSend.Align = AlTop;
 BtnSend.Margins.Right = 100;
 BtnSend.Margins.Left = 100;
 BtnSend.Margins.Bottom = 8;
 BtnSend.Margins.Top = 8;
 BtnSend.clProSettings.TextSettings.Font.Style = [fsBold];
 BtnSend.clProSettings.IsRound = True;
 BtnSend.clProSettings.RoundHeight = 2;
 BtnSend.clProSettings.RoundWidth = 2;
 BtnSend.clProSettings.BorderColor = clAlphaColor.clHexToColor('#808080');
 BtnSend.clProSettings.BorderWidth = 2;
 BtnSend.SetclProSettings(BtnSend.clProSettings);
 MyForm.AddNewEvent(BtnSend,tbeOnClick,'BtnSendClick');

 MyForm.Run;

}
```

10.6.2. TclWebBrowser

Clomosy platformu üzerinde bir web tarayıcısını entegre etmek için *TclWebBrowser* bileşeni kullanılmaktadır. *AddNewWebBrowser* fonksiyonu ile bir Web Browser oluşturulmaktadır. Bu işlev iki parametre almaktadır. İlk parametre, tanımlanacak olan nesnenin hangi bileşen içerisinde konumlanacağını belirlemektedir. İkinci parametre, tanımlanan nesnenin ismini temsil etmektedir.

Oluşturulan değişkenin *Navigate* özelliği üzerinden tarayıcıyı belirtilen URL'e yönlendirmektedir. Bu yöntem, belirli bir web sayfasını yüklemek istediğinizde kullanılmaktadır.

```
Var
 anaForm:TclForm;
 xWeb:TclWebBrowser;

{
 anaForm = TclForm.Create(Self);
 xWeb = anaForm.AddNewWebBrowser(anaForm,'xWeb');
 xWeb.Align = alClient;
 xWeb.Navigate('https://www.google.com');
 anaForm.Run;
}
```

10.6.3. TclDeviceManager

TclDeviceManager bileşeni, cihazlarla etkileşimde bulunmak için kullanılmaktadır. Bu bileşen, cihazların özelliklerini ve işlevlerini kontrol etmenizi sağlamaktadır.

Özellikler şunları içerir:

- Titreşim
- Pil seviyesi
- Telefon rehberine erişim

TclDeviceManager sınıfına ait bir değişken tanımlanır. Nesneyi oluşturmak için *Create* işlemi yapılır.

Titreşim için nesne özelliği olan *Vibrate()* kullanılır. Bir parametre alır ve bu parametre kaç saniye titretileceğinin ayarlanmasını sağlar. Parametre milisaniye cinsinden yazılır. Cihaz batarya durumunu öğrenmek için *BatteryStatus* özelliği ve son olarak rehber erişimi için *ContactsList* özelliği kullanılır.

Aşağıda bu özelliklerin kullanımına dair örnek verilmiştir.

```
var
  anaForm : TclForm;
  MyDevice:TclDeviceManager;
  i : Integer;
  Memo1 : TclMemo;

void ListeGetir;
{
   try

   for (i = 0 to MyDevice.ContactsList.Count - 1)
     {
      Memo1.Lines.Add('Adı: ' + Clomosy.StringListItemString(MyDevice.ContactsList,i));
      Memo1.Lines.Add('---');
     }
   finally
    MyDevice.ContactsList.Free;
   }
  }

{
  anaForm = TclForm.Create(Self);
  Memo1 = anaForm.AddNewMemo(anaForm,'Memo1','');
  Memo1.Align = alClient;
  Memo1.Margins.Left= 10;
  Memo1.Margins.Right= 10;
  Memo1.Margins.Top= 10;
  Memo1.Margins.Bottom= 10;
```

```
  Memo1.ReadOnly = True;
  Memo1.TextSettings.WordWrap = True;

  MyDevice=TclDeviceManager.Create;
  ShowMessage(MyDevice.BatteryStatus);
  MyDevice.Vibrate(500);
  MyDevice.GetAddressBookContacts('ListeGetir');
  ShowMessage('Liste Adı: '+MyDevice.CallOnAfterListProcName);

  anaForm.Run;
}
```

10.6.4. TclMqtt

MQTT (Message Queuing Telemetry Transport), düşük bant genişliği ve düşük enerji tüketimi gibi özelliklere odaklanan bir mesajlaşma protokolüdür. Özellikle Internet of Things (IoT) uygulamalarında kullanılmaktadır. Protokol, birçok cihazın birbirleriyle veri paylaşmasını ve iletişim kurmasını sağlamak üzere tasarlanmıştır.

Clomosy üzerinden Mqtt teknolojisi TclMqtt bileşeni ile sağlanmaktadır. Bu teknoloji ile mesajlaşma, bir cihaza uzaktan erişim sağlama gibi programlar yazılabilir.

AddNewMQTTConnection fonksiyonu ile bir MQTT nesnesi oluşturulmaktadır. Bu işlev iki parametre almaktadır. İlk parametre, tanımlanacak olan nesnenin hangi bileşen içerisinde konumlanacağını, ikinci parametre ise tanımlanan nesnenin ismini temsil etmektedir.

MQTT üzerinden haberleşmeyi sağlayabilmek için event özellikleri olan *tbeOnMQTTStatusChanged* ve *tbeOnMQTTPublishReceived* kullanılmaktadır. *tbeOnMQTTStatusChanged* işlemi, MQTT bağlantısının durumu değiştiğinde, *tbeOnMQTTPublishReceived* olayı ise MQTT mesajı alındığında meydana gelen bir olaydır.

Ağ üzerinden haberleşerek verilerin aktarıldığı bir bileşen olduğu için MQTT aracısına veya bir konuya yönlendirilen mesajlar *Channel* özelliği üzerinden alınmaktadır.

Ağdan gelen mesajlara erişmek için *ReceivedMessage* kullanılmaktadır. Aşağıdaki örnekte nasıl kullanıldığı ele alınmıştır.

```
var
 Form1 : TCLForm;
 MyMQTT : TclMQTT;
 gonderBtn: TclProButton;
 durumlbl: TclLabel;
 msjgonderMemo, msjListesi: TclMemo;
 msjKutusuPnl, anaPnl : TclProPanel;
 msjLyt : TclLayout;

 void MemMsgEnter;
 {
```

```
msjLyt.Align=alMostTop;
 }
 void MyMQTTStatusChanged;
 {
  if (MyMQTT.Connected )
  {
  durumlbl.Text = 'Bağlandı' ;
  durumlbl.TextSettings.FontColor = clAlphaColor.clHexToColor('#0d7d0b');
  }
  Else
  {
  durumlbl.Text = 'Bağlandı koptu!';
  durumlbl.TextSettings.FontColor = clAlphaColor.clHexToColor('#7a1209');

  }
 }

 void MyMQTTPublishReceived;
 {
  if(MyMQTT.ReceivedAlright)
  {
    msjListesi.Lines.Add('                  ' + MyMQTT.ReceivedMessage);
  }
 }
void BtnSendClick;
 {
  MyMQTT.Send(Clomosy.AppUserDisplayName+' : '+msjgonderMemo.Text);
  msjListesi.Lines.Add('Sen: '+msjgonderMemo.Text);
  msjgonderMemo.Text = ' ';
  msjLyt.Align=alBottom;
  msjListesi.setFocus;
 }
{
 Form1 = TCLForm.Create(Self);

 anaPnl=Form1.AddNewProPanel(Form1,'anaPnl');
 anaPnl.Align = AlClient;
 anaPnl.Margins.Top = 10;
 anaPnl.Margins.Bottom = 10;
 anaPnl.Margins.Right = 10;
 anaPnl.Margins.Left = 10;
 anaPnl.clProSettings.IsRound = True;
 anaPnl.clProSettings.RoundHeight = 10;
 anaPnl.clProSettings.RoundWidth = 10;
 anaPnl.clProSettings.BorderColor = clAlphaColor.clHexToColor('#5f00bf');
 anaPnl.clProSettings.BorderWidth = 3;
 anaPnl.SetclProSettings(anaPnl.clProSettings);

 durumlbl= Form1.AddNewLabel(anaPnl,'durumlbl','--');
```

```
 durumlbl.Align = alTop;
 durumlbl.Visible = True;
 durumlbl.Margins.Left = 10;
 durumlbl.StyledSettings = ssFamily;
 durumlbl.TextSettings.Font.Size = 13;

 MyMQTT = Form1.AddNewMQTTConnection(anaPnl,'MyMQTT');
 Form1.AddNewEvent(MyMQTT,tbeOnMQTTStatusChanged,'MyMQTTStatusChanged');

Form1.AddNewEvent(MyMQTT,tbeOnMQTTPublishReceived,'MyMQTTPublishReceive
d');

 MyMQTT.Channel = 'chat';//project guid + channel
 MyMQTT.Connect;

 msjListesi= Form1.AddNewMemo(anaPnl,'msjListesi','');
 msjListesi.Align = alClient;
 msjListesi.Margins.Top = 10;
 msjListesi.Margins.Bottom = 10;
 msjListesi.Margins.Left = 10;
 msjListesi.Margins.Right = 10;
 msjListesi.TextSettings.WordWrap = True;
 msjListesi.ReadOnly = True;

 msjLyt = Form1.AddNewLayout(anaPnl,'msjLyt');
 msjLyt.Align=alBottom;
 msjLyt.Height = 90;
 msjLyt.Margins.Bottom = 5;
 msjLyt.Width = 400;
 msjLyt.Margins.Left = 10;
msjLyt.Margins.Right = 10;

 gonderBtn = Form1.AddNewProButton(msjLyt,'gonderBtn','');
 gonderBtn.Align = AlRight;
 gonderBtn.Width = 80;
 gonderBtn.Margins.Top = 15;
 gonderBtn.Margins.Bottom = 15;
 gonderBtn.Margins.Left = 15;
 gonderBtn.clProSettings.BorderColor = clAlphaColor.clHexToColor('#ffffff');
 gonderBtn.SetclProSettings(gonderBtn.clProSettings);

 Form1.SetImage(gonderBtn,'https://clomosy.com/educa/send.png');
 Form1.AddNewEvent(gonderBtn,tbeOnClick,'BtnSendClick');

 msjKutusuPnl=Form1.AddNewProPanel(msjLyt,'msjKutusuPnl');
 msjKutusuPnl.Align = AlLeft;
 msjKutusuPnl.Width = 240;
```

```
 msjKutusuPnl.Margins.Top = 20;
 msjKutusuPnl.Margins.Bottom = 20;
 msjKutusuPnl.Margins.Left = 10;
 msjKutusuPnl.clProSettings.IsRound = True;
 msjKutusuPnl.clProSettings.RoundHeight = 15;
 msjKutusuPnl.clProSettings.RoundWidth = 15;
 msjKutusuPnl.clProSettings.BorderColor = clAlphaColor.clHexToColor('#bf00bf');
 msjKutusuPnl.clProSettings.BorderWidth = 1;
 msjKutusuPnl.SetclProSettings(msjKutusuPnl.clProSettings);

 msjgonderMemo= Form1.AddNewMemo(msjKutusuPnl,'msjgonderMemo','');
 msjgonderMemo.Align = alClient;
 msjgonderMemo.Margins.Left = 5;
 msjgonderMemo.Margins.Right = 5;
 msjgonderMemo.Margins.Bottom = 5;
 msjgonderMemo.Margins.Top = 5;
 msjgonderMemo.TextSettings.WordWrap = True;
 Form1.AddNewEvent(msjgonderMemo,tbeOnEnter,'MemMsgEnter');

 Form1.SetFormBGImage('https://clomosy.com/educa/bg2.png');
 Form1.Run;
}
```

10.7. Tasarım Tipi Bileşenler

Uygulama arayüzünü daha etkin kullanabilmek, bazı kontrollerin şekil ve durumunu son kullanıcının tasarım yeteneğine bırakmak amacıyla oluşturulan yardımcı bileşenlerdir.

10.7.1. TclExpander

Bu bileşen, bir grup kontrolü içinde görünen bir "açılır/kapanır" panel veya bölme oluşturmaktadır. Kullanıcılar, bu açılır/kapanır bölme üzerindeki bir düğmeye tıklayarak içeriği gizleme veya görüntüleme işlemini gerçekleştirmektedir.

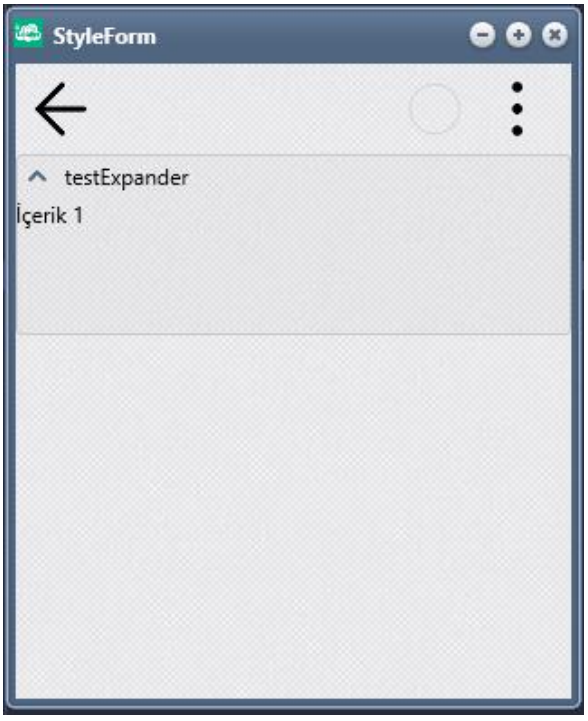
TclExpander bileşeni, bir bölgenin genişletilmiş (açık) veya daraltılmış (kapalı) durumunu kontrol etmek için kullanılmaktadır. Bu, bir kullanıcı arayüzünde belirli bir bilgi veya kontrol grubunu gizlemek ve gerektiğinde göstermek için kullanışlıdır.

AddNewExpander fonksiyonu kullanılarak, *TclExpander* oluşturulmaktadır. Bu işlev üç parametre almaktadır. İlk parametre tanımlanacak olan 'açılır panel'in hangi bileşen içerisinde konumlanacağını belirler. İkinci parametre, tanımlanan açılır panel ismini temsil etmektedir. Üçüncü parametre ise 'açılır panel'e içerik adlandırması yapmaktadır.

Aşağıdaki örnekte, formda oluşturulan expander bileşeninin içine TclLabel nesnesi konularak basit bir gösterim yapılmıştır.

```
var
  anaForm:TclStyleForm;
  testExpander:TclExpander;
  tstLbl:TclLabel;

{
  anaForm  = TclStyleForm.Create(Self);
  anaForm.clSetStyle(anaForm.LightSB);

  testExpander = anaForm.AddNewExpander(anaForm,'testExpander','');
  testExpander.Align  = alTop;
  testExpander.Height  = 100;
  tstLbl = anaForm.AddNewLabel(testExpander,'tstLbl','My Expander Label Caption');
  tstLbl.Align  = alTop;

  anaForm.Run;

}
```

Aşağıda örneğin görüntüsü verilmiştir.



Resim 57

10.7.2. TclMenuFrame

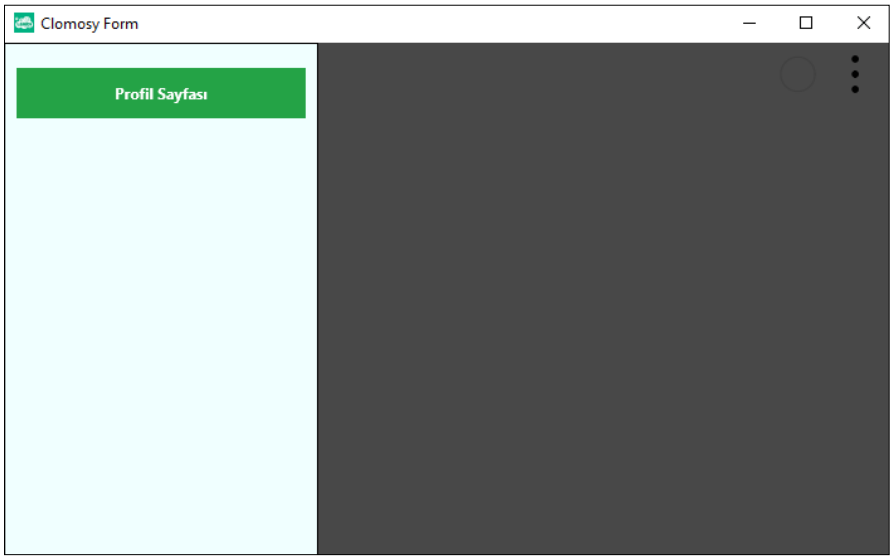
TclMenuFrame, Clomosy platformunun sunmuş olduğu bir menü bileşenidir. Uygulamada menü seçeneklerinin oluşturulmasına yardımcı olmaktadır. Bir *TclMenuFrame* oluşturmak için

AddNewMenuFrame fonksiyonu kullanılmaktadır. Bu işlev; tanımlanacak olan menü nesnesinin hangi bileşen içinde konumlanacağını belirleyen ve tanımlanan menü nesnesinin ismini temsil eden iki parametre almaktadır.

'ClMenuPosition' özelliği kullanılarak menü çerçeve konumu ayarlanmaktadır. Aşağıdaki örnekte bir nesne oluşturulmuş ve içerisine bir tane buton konulmuştur.

```
Var
  anaForm : TclForm;
  TstSideMenu : TClMenuFrame;
  profilBtn: TClProButton;
{
  anaForm = TclForm.Create(Self);
  TstSideMenu = anaForm.AddNewMenuFrame(anaForm,'TstSideMenu');
  TstSideMenu.Align = alContents;
  TstSideMenu.MenuBar.Width = 250;
  TstSideMenu.ClMenuPosition = clLeft;   //Default clRight
  profilBtn   =   anaForm.AddNewProButton(TstSideMenu.VertScrollBox,'profilBtn','Profil
Sayfası');
  profilBtn.Align = AlTop;
  profilBtn.Height = 40;
  profilBtn.Margins.Top = 10;
  profilBtn.clProSettings.FontColor = clAlphaColor.clHexToColor('#ffffff');
  profilBtn.clProSettings.TextSettings.Font.Style = [fsBold];
  profilBtn.clProSettings.BackgroundColor = clAlphaColor.clHexToColor('#24a346');
  profilBtn.clProSettings.FontVertAlign = palcenter;
  profilBtn.SetclProSettings(profilBtn.clProSettings);
  anaForm.Run;
}
```

Aşağıda örneğin görüntüsü verilmiştir.



Resim 58

10.8. Grafik ve Fotoğraf Gösterim Bileşenleri

Kullanıcı arayüzünün daha görsel hale getirilmesi, özel fotoğrafların listelenmesi, bir konu hakkında grafik gösteriminin yapılması gibi farklı özelliklerin kullanılmasına dair bileşenlerdir.

10.8.1. TclImage

Bu bileşen, bir form veya başka bir konteyner içinde resimleri görüntülemektedir. TclImage bileşeni, farklı grafik dosya türlerini destekler ve bir resmi görüntülemek için kullanıcıya olanak tanımaktadır.

Bir görsel nesnesi oluşturmak için *AddNewImage* fonksiyonu kullanılır. Bu işlev iki parametre almaktadır. İlk parametre tanımlanacak olan görsel nesnesinin hangi bileşen içerisinde konumlanacağını belirler. İkinci parametre tanımlanan görsel nesnesinin ismini temsil eder.

TclImage nesnesine görsel eklemek için formun *setImage* özelliği kullanılmaktadır. *MultiResBitmap* özelliğinin *Clear* metodu ise bileşene eklenen resmin silinmesi ve bileşenin tamamen boşaltılmasını sağlamaktadır.

Aşağıdaki örnekte bir form üzerine bir resim tanımlanmıştır. Resmin üzerine tıklandığında bileşen üzerinden resim silinir ve yeni bir resim eklenmektedir.

```
var
  anaForm:TclForm;
  testImg : TclImage;

void resmiSil;
{
  testImg.MultiResBitmap.Clear;
  anaForm.setImage(testImg,'https://clomosy.com/demos/balloon2.png');
}
{
  anaForm = TclForm.Create(Self);
  testImg= anaForm.AddNewImage(anaForm,'testImg');
  testImg.Align = alCenter;
  testImg.Height = 250;
  testImg.Width = 300;

  anaForm.setImage(testImg,'https://clomosy.com/demos/balloon3.png');
  anaForm.AddNewEvent(testImg,tbeOnClick,'resmiSil');
  anaForm.Run;

}
```

10.8.2. TclProImage

TclProImage gibi bir form veya başka bir konteyner içinde resimleri görüntülemek için kullanılmaktadır. *TclImage* özelliklerinin hepsini kapsar. Bunun dışında 'ProImage'ın; arka plan rengi, resim üzerine metin ekleme, kenarlık rengi vb. özel kullanımları da bulunmaktadır.

AddNewProImage fonksiyonu ile profesyonel resim oluşturulmaktadır. Bu işlev iki parametre almaktadır. İlk parametre, tanımlanacak olan resim nesnesinin hangi bileşen içerisinde konumlanacağını belirler. İkinci parametre, tanımlanan resim nesnesinin ismini temsil eder. *TclProImage* nesnesine görsel eklemek için standart görsel ekleme dışında clProSettings yöntemi ile de eklenebilir.

```
testImg.clProSettings.PictureSource = 'https://clomosy.com/demos/balloon3.png';
testImg.clProSettings.PictureAutoFit = True;
```

TclProImage kullanımında bir resmi değiştirmek istediğinizde bileşen içerisinde silme işlemi (*MultiResBitmap.Clear*) yapılmaz. Yeniden resim atama işlemi yapılması yeterlidir. *TclImage* konu başlığı altında yapılan örneği bir de *TclProImage* üzerinden programlayalım.

```
var
  anaForm:TclForm;
  testImg : TclProImage;

void resmiDegistir;
{
  testImg.clProSettings.PictureSource = 'https://clomosy.com/demos/balloon2.png';
  testImg.SetclProSettings(testImg.clProSettings);
}
{
  anaForm = TclForm.Create(Self);
  testImg= anaForm.AddNewProImage(anaForm,'testImg');
  testImg.clProSettings.PictureSource = 'https://clomosy.com/demos/balloon3.png';
  testImg.clProSettings.PictureAutoFit = True;
  testImg.SetclProSettings(testImg.clProSettings);
  anaForm.AddNewEvent(testImg,tbeOnClick,'resmiDegistir');
  anaForm.Run;
}
```

10.8.3. TclChart

Grafik ve çizimler oluşturmak için kullanılan bu bileşen, bir form üzerinde veya başka bir konteyner içinde de kullanılma özelliğine sahiptir. *TclChart* bileşeni, destekleyerek çeşitli veri görselleştirmelerinin oluşturulmasını sağlamaktadır. Bu grafik türleri arasında çizgi grafikleri, sütun grafikleri, pasta grafikleri ve alan grafikleri bulunmaktadır.

AddNewChart fonksiyonu kullanılarak oluşturulan bileşen, üç parametre almaktadır. İlk parametre, tanımlanacak olan grafiğin hangi bileşen içerisinde konumlanacağını belirler. İkinci parametre, tanımlanan grafiğin ismini, üçüncü parametre ise başlığını temsil etmektedir.

TclChart bileşenine veri eklemenin 2 farklı yöntemi bulunmaktadır. Json yapısı ile veri eklenebilmesi için *clLoadDataFromJSONStr* prosedürü kullanılır veya veri seti üzerinden veri alınarak yani *clLoadDataFromDataset* prosedürü kullanılarak veri eklenebilir.

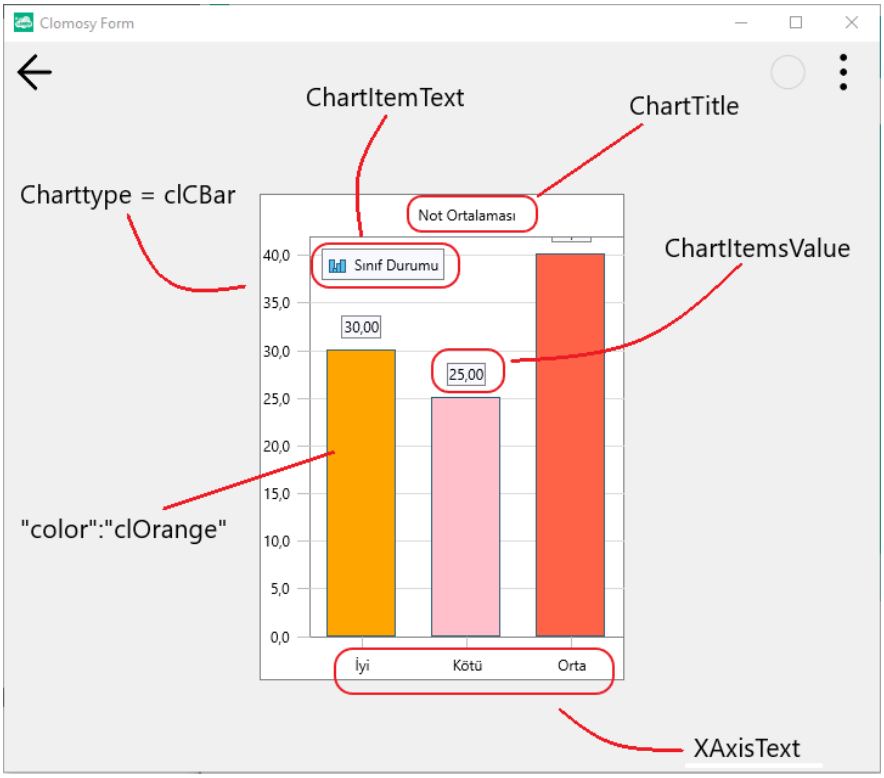
Grafik üzerinde x ekseninde karşılaştırılacak değerlerin tanımlanması için *XAxisText*, y ekseni üzerindeki değerlerin derecelerinin yazıldığı özellik ise *ChartItemsValue*'dir. Bu derecelerin başlıklarını belirtmek için *ChartItemText* özelliği kullanılmaktadır. Grafik bileşeninin içerisindeki grafik serilerinin (series) kenar boşluklarını tanımlamak için *SeriesMargins.Top* özelliğine değer atılması gerekmektedir.

Grafik başlığının tanımlanması için grafik nesnesinin *ChartTitle* özelliği kullanılmaktadır.

Dört adet grafik görünümü mevcuttur. Grafik gösterimini değiştirmek için *Charttype* parametresi kullanılır. Grafik görünümü özellikleri aşağıdaki tabloda yer almaktadır.

Grafik Görünümü Tanımı	Özelliği
clCLine	Çizgi grafik
clCBar	Çubuk grafik
clCPie	Pasta grafik
clCSizedPie	Boyutlandırılmış pasta grafik

Görünüm aşağıdaki gibidir.



Grafikte, değer renklerini belirlerken 'color' olarak tanımlama yapılmaktadır. Json yapısındaki örneği ise aşağıda verilmiştir.

```
veriJSONS = '[{"NotTipi" : "Kötü","Yuzde": 15,"color":"clPink"}]';
```

Clomosy platformunda grafikte belirli renkler verilmektedir. Aşağıda kullanılabilecek renkler yer almaktadır.

- clGreen
- clRed
- clYellow
- clBlue
- clYellowGreen
- clAzure
- clTomato
- clPurple
- clPink
- clGold
- clPeru
- clOrange

Bu özelliklerin kullanımını bir örnek üzerinde ele alalım.

```
var
  anaForm:TclForm;
  testChart : TClChart;
  veriJSON : String;

{
  anaForm = TclForm.Create(Self);
  testChart = anaForm.AddNewChart(anaForm,'testChart','Not Ortalaması');
  veriJSON = '[{"NotTipi": "İyi","Yuzde": 30,"color":"clOrange"},
  {"NotTipi" : "Kötü","Yuzde": 25,"color":"clPink"},
  {"NotTipi" : "Orta","Yuzde": 40,"color":"clTomato"}]';
   testChart.SeriesMargins.Top = 20;

  testChart.Charttype = clCBar;
  testChart.XAxisText = 'NotTipi';
  testChart.ChartItemText = 'Sınıf Durumu';
  testChart.ChartItemsValue = 'Yuzde';
  testChart.ChartTitle = 'Not Ortalaması';
  testChart.clLoadDataFromJSONStr(veriJSON);
  anaForm.Run;

}
```

10.8.4. TclQRCodeGenerator

Clomosy platformunun sunmuş olduğu QR kod üretebilen bir bileşendir. *AddNewQRCodeGenerator* fonksiyonu kullanılarak oluşturulan bileşen, üç parametre almaktadır. İlk parametre, tanımlanacak olan nesnenin hangi bileşen içerisinde konumlanacağını belirler. İkinci parametre, tanımlanan nesnenin ismini, üçüncü parametre ise başlığını temsil etmektedir. QR kod içerisine bir verinin yazılması için nesnenin *Text* özelliğine veri ataması yapılmalıdır. Örneğin; QR kod okutulduğunda içerisinden saat verisi gelsin isteniyorsa aşağıdaki gibi yapılabilir.

```
QRGen.Text = FormatDateTime('ddmmyy hhnn', Now);
```

TclQRCodeGenerator bileşenin event özelliği mevcuttur. Bu olay tetikleyici *tbeOnGetQRCode* olarak adlandırılmaktadır. Bu olay tetikleyicisi yeni bir QR kod üretildiğinde tetiklenen bir işlemdir.

Aşağıda örnek bir kullanımı mevcuttur. QR kod okutulduğu an Clomosy doküman sitesine yönlendirecektir.

```
var
  anaForm :TclForm;
  QRGen :  TClQRCodeGenerator;

{
  anaForm = TclForm.Create(Self);

  QRGen= anaForm.AddNewQRCodeGenerator(anaForm,'QRGen','QRKod');
  QRGen.Height = 250;
  QRGen.Width = 250;
  QRGen.Align = alCenter;
  QRGen.Text = 'https://www.docs.clomosy.com';
  //anaForm.AddNewEvent(QRGen,tbeOnGetQRCode,'');

  anaForm.Run;

}
```

Resim 59

10.9. Akış ve BLOB Bileşenleri

10.9.1. TclMemoryStream

TclMemoryStream, verilerini dinamik bellekte saklayan bir akıştır. Verileri, dosya benzeri erişim özellikleriyle geliştirilmiş dinamik bir bellek arabelleğinde depolamak için *TclMemoryStream'i* kullanılır. *TclMemoryStream*, dinamik bir bellek arabelleğini yönetmek için yöntemler ve özellikler sunarken bir akış nesnesinin genel G/Ç yeteneklerini sağlar.

Bellek akışları, bilgiyi tutabilen, aynı zamanda onu okuyabilen veya başka bir depolama ortamına yazabilen aracı nesneler olarak kullanışlıdır. Akışların içeriğini karşılaştırmak veya daha az erişilebilir bir ortamda depolanan verileri değiştirmek için kullanışlı bir format sağlarlar.

Artıları:

- Hızlı Erişim: Bellek erişimi, dosya I/O işlemlerinden daha hızlıdır.
- Kolay Manipülasyon: Geçici veri depolama ve manipülasyon için idealdir.
- Disk I/O Yok: Disk erişimine gerek olmadığı için sık okuma/yazma gerektiren işlemlerde daha hızlıdır.

Eksileri:

- Bellek Kullanımı: Mevcut sistem belleği miktarıyla sınırlıdır. Büyük veriler önemli miktarda bellek tüketebilir.

- Geçici Depolama: Veriler geçici bellekte saklandığından uygulama kapatıldığında kaybolur.

Öncelikle, *TclMemoryStream* sınıfından bir değişken oluşturulur ve ardından forma yeni bir nesne oluşturulmalıdır.

```
var
   MemStream: TclMemoryStream;
   Form1:TCLForm;
{
  Form1 = TCLForm.Create(Self);
  MemStream = TCLMemoryStream.Create;
  Form1.Run;
}
```

Bellekteki veriye erişmek için çeşitli yöntemler kullanılabilir. Bu yöntemler arasında, veriyi Base64 formatında döndürme için *AsBase64* fonksiyonu kullanılır. Örnek olarak alınan veri bir TclMemo içerisine Base64 string yapısına dönüştürülerek aktarılmaktadır.

```
noteMemo.Lines.Text = MemStream.AsBase64;
```

Bunun dışında bellek akışının boyutunu belirleme, akışı temizleme ve bellek alanını serbest bırakma gibi işlemler bulunmaktadır. Ayrıca, TCLMemoryStream bileşeni, görüntü verilerini saklamak ve taşımak için de kullanılabilir. Görüntü verilerini *TCLMemoryStream* nesnesine kaydedebilir ve daha sonra bu veriyi bir *TclImage* bileşeni aracılığıyla yükleyebilirsiniz. Bu şekilde, bellek akışı ve görüntü bileşenleri arasında veri transferi sağlanır ve görüntülerin bellekte saklanması ve işlenmesi kolaylaşır.

Aşağıdaki örnekte, Base64 kodlu bir görüntünün, bir görüntü (TclImage) ve bir not bileşeni (TclMemo) içeren bir form üzerinde kodunun çözülmesini ve görüntülenmesini gösterir. NoteMemoOnClick prosedürü, nottaki Base64 kodlu metni bir bellek akışına dönüştürür ve ardından bu akışı, görüntüyü SourceImg bileşenine yüklemek için kullanır.

```
var
   Form1:TCLForm;
   SourceImg : TclImage;
   noteMemo : TclMemo;
   btn1 : TclButton;
 void noteMemoOnClick;
 var
   LMemStream:TCLMemoryStream;
 {
   LMemStream = TCLMemoryStream.Create;
   LMemStream = Clomosy.Base64ToStream(noteMemo.Lines.Text);
   SourceImg.Bitmap.LoadFromStream(LMemStream);
}
```

```
{
  Form1 = TCLForm.Create(Self);
  Form1.AddAssetFromUrl('https://clomosy.com/educa/ok.png');
  noteMemo = Form1.AddNewMemo(Form1,'noteMemo','---');
  noteMemo.Align = alMostTop;
  noteMemo.Height = 200;
  noteMemo.StyledSettings = ssFamily;
  noteMemo.TextSettings.WordWrap = True;
  noteMemo.Lines.Text =
Clomosy.FileToBase64(clPathCombine('ok.png',Clomosy.AppFilesPath));

  SourceImg = Form1.AddNewImage(Form1,'SourceImg');
  SourceImg.Align = alClient;
  Form1.setImage(SourceImg,'https://clomosy.com/educa/hint.png');
  btn1 = Form1.AddNewButton(Form1,'btn1','Yükleme');
  btn1.StyledSettings = ssFamily;
  btn1.TextSettings.Font.Size = 20;
  btn1.TextSettings.FontColor = clAlphaColor.clHexToColor('#8a067c');
  btn1.Align = alBottom;
  Form1.AddNewEvent(btn1,tbeOnClick,'noteMemoOnClick');

  Form1.Run;
 }
```

10.9.2. TclFileStream

TclFileStream, uygulamaların diskteki bir dosyadan okumasına ve bu dosyaya yazmasına olanak tanır. Disk dosyalarındaki bilgilere erişmek için *TclFileStream* kullanılır. *TclFileStream* adlandırılmış bir dosyayı açacak ve bu dosyadan okuma veya yazma yöntemleri sağlayacaktır.

Artıları:

- Büyük Veri İşleme: Yalnızca disk alanıyla sınırlı olan çok büyük veri boyutlarını işleyebilir.
- Kalıcılık: Veriler kalıcıdır ve uygulama kapatıldıktan sonra bile kullanılabilir durumda kalır.

Eksileri:

- Daha Yavaş Erişim: Dosya I/O işlemleri, disk erişim gecikmelerinden dolayı genellikle bellek işlemlerinden daha yavaştır.
- Disk I/O: Diskten okumayı ve diske yazmayı içerir; bu daha yavaş olabilir ve kaynak açısından yoğun olabilir.

Öncelikle, *TclFileStream* sınıfından bir değişken oluşturulur.

```
FileStream: TCLFileStream;
```

Ardından forma yeni bir nesne oluşturulmalıdır. Form üzerinde yeni bir *TclFileStream* nesnesi oluştururken iki parametre alır. İlk parametre dosya yolunu, ikinci parametre ise dosyanın nasıl açılacağını belirtir. Örnekte belirtilen dosyayı oluşturur ve yazma modunda (fmCreate) açar.

Dosyanın yazma modunda başka açılma yöntemleride bulunmaktadır. Bunlar;

Özellik	Kullanım Amacı
fmOpenRead	Dosya okuma modunda açılır. Dosya sadece okunabilir.
fmOpenWrite	Dosya yazma modunda açılır. Dosya üzerine yazılır, varsa içeriği silinir.
fmOpenReadWrite	Dosya okuma ve yazma modunda açılır. Dosya içeriği değiştirilebilir.
fmExclusive	Dosya başka bir işlem tarafından kullanılamaz. Diğer işlemler dosyayı açamaz.
fmShareExclusive	Dosya birden fazla işlem tarafından açılabilir ancak bu sırada diğer işlemler dosyayı açamaz.
fmShareDenyWrite	Dosya birden fazla işlem tarafından açılabilir ancak bu sırada diğer işlemler dosyayı yazamaz.
fmShareDenyNone	Dosya birden fazla işlem tarafından açılabilir ve bu sırada diğer işlemler dosyayı okuyabilir veya yazabilir.
fmClosed	Dosya kapalı durumda açılır. (Kullanımı sınırlıdır)
fmInput	Giriş akışı olarak açılır.
fmOutput	Çıkış akışı olarak açılır.
fmInOut	Giriş ve çıkış akışı olarak açılır.

Kullanımı;

```
FileStream = TCLFileStream.Create(clPathCombine('ok.png',Clomosy.AppFilesPath), fmCreate);
```

Veriyi Base64 formatında döndürme için *AsBase64* fonksiyonu kullanılır. Örnek olarak alınan veri bir *TclMemo* içerisine Base64 string yapısına dönüştürülerek aktarılmaktadır.

```
noteMemo.Lines.Text = FileStream.AsBase64;
```

Örnekte, bir form oluşturulur ve üzerine bir resim, bir metin alanı ve bir düğme eklenir. *FileStreamExample* adlı prosedür önce görüntüyü bir dosyaya kaydeder (fmCreate modunda), ardından aynı görüntüyü dosyadan yükler (fmOpenRead modunda) ve son olarak görüntünün Base64 kodunu bir metin alanına yerleştirir.

```
var
   Form1:TCLForm;
   SourceImg : TclImage;
   noteMemo : TclMemo;
   btn1 : TclButton;

void FileStreamExample;
var
  FileStream: TCLFileStream;
{
  // Save Img To File
  FileStream = TCLFileStream.Create(clPathCombine('ok.png',Clomosy.AppFilesPath),
fmCreate);
  try
    SourceImg.Bitmap.SaveToStream(FileStream);
  finally
    FileStream.Free;
  }
  // Load Img From File
  FileStream = TCLFileStream.Create(clPathCombine('ok.png',Clomosy.AppFilesPath),
fmOpenRead);
try
    SourceImg.Bitmap.LoadFromStream(FileStream);
    noteMemo.Lines.Text = FileStream.AsBase64;
  finally
    FileStream.Free;
  }
}
{
  Form1 = TCLForm.Create(Self);
  Form1.AddAssetFromUrl('https://clomosy.com/educa/ok.png');
  noteMemo = Form1.AddNewMemo(Form1,'noteMemo','---');
  noteMemo.Align = alMostTop;
  noteMemo.Height = 200;
  noteMemo.StyledSettings = ssFamily;
  noteMemo.TextSettings.WordWrap = True;

  SourceImg = Form1.AddNewImage(Form1,'SourceImg');
  SourceImg.Align = alClient;
  Form1.setImage(SourceImg,'https://clomosy.com/educa/hint.png');

  btn1 = Form1.AddNewButton(Form1,'btn1','Upload');
  btn1.StyledSettings = ssFamily;
  btn1.TextSettings.Font.Size = 20;
  btn1.TextSettings.FontColor = clAlphaColor.clHexToColor('#8a067c');
  btn1.Align = alBottom;
  Form1.AddNewEvent(btn1,tbeOnClick,'FileStreamExample');
  Form1.Run;
}
```

10.9.3. TclBlobField

TclBlobField, veri kümesindeki ikili büyük nesneye (BLOB - İkili Büyük Nesne) referansı tutan bir alanı temsil eder.

TclBlobField, ikili büyük nesne (BLOB - İkili Büyük Nesne) alanlarında ortak olan temel davranışı kapsar. BLOB alanları, isteğe bağlı uzunluktaki ham ikili verileri içeren veritabanı alanlarıdır. BLOB alanları farklı, isteğe bağlı olarak büyük veri türlerini temsil edebilir. Bu veri türleri ikili verinin başlığında ayırt edilir.

BLOB alanları metin, resim, ses veya video gibi büyük ikili verilerin depolanmasına olanak tanır.

Öncelikle, *TclBlobField* sınıfından bir değişken oluşturulur.

```
BlobField : TclBlobField;
```

TclBlobField nesnesine veritabanından atama yapabilmek için aşağıdaki gibi veritabanından bir veri alınmalıdır.

```
BlobField = qry.FieldByName('DOCUMENT_FILE') as TclBlobField;
```

Aşağıdaki örnek ile nasıl kullanılır bakalım.

Örnek kod bir SQL Server veritabanına bağlanır, bir tabloda bir sorgu yürütür ve bir BLOB (İkili Büyük Nesne) alanından görüntü verilerini yükler. Yüklenen görüntü verilerini bir akışa (TclMemoryStream) kaydeder, ardından bu verileri bir görsel bileşene (TclImage) yükler ve görüntüler. Ayrıca Base64 formatındaki görüntü verilerini panoya (setClipBoard) kopyalar.

```
var
  Server, User, Password,DB: String;
  Port: Integer;
  Form1:TCLForm;
  qry : TClSqlQuery;
  BlobStream: TclMemoryStream;
  BlobField: TclBlobField;
  Img1 : TclImage;
  Base64String: string;

 void MainForm;
 {
  Form1 = TCLForm.Create(Self);
  Img1 = Form1.AddNewImage(Form1,'Img1');
  Img1.Align = alClient;

  qry = TClSqlQuery.Create(Nil);
```

```
  try
    try
      qry.Connection = Clomosy.DBSQLServerConnection;
      qry.Sql.Text = 'SELECT TOP(1) DOC_FILE FROM ATBLDOCS';
      qry.Open;
      if (qry.Found)
      {
        BlobField = qry.FieldByName('DOC_FILE') as TclBlobField;
        BlobStream = TclMemoryStream.Create;
        BlobField.SaveToStream(BlobStream); // BLOB verileri akışa yüklenir
        BlobStream.Position = 0;
        Base64String = Clomosy.StreamToBase64(BlobStream);
        Clomosy.setClipBoard(Base64String);

      Img1.Bitmap.LoadFromStream(BlobStream); // Görüntü akıştan BlobStream nesnesine
yüklenir
        BlobStream.Free;
      }
    except
      ShowMessage('[01] Exception Class: '+LastExceptionClassName+' Exception Message:
'+LastExceptionMessage);
    }
  finally
    qry.Free;
  }
  Form1.Run;
 }

 {
  try
    Server = '192.168.X.XX';
    User = 'xx';
    Password = 'xxxx';
    DB = 'CLS23';
    Port = 1433;
    Clomosy.EventLog = True;
    if not (Clomosy.DBSQLServerConnect('SQL Server', Server, User, Password, DB, Port))
    {
      ShowMessage('DATABASE CONNECT ERROR');
      Exit;
    }
    MainForm;
  except
    ShowMessage('Connection Error '+'Exception Class: '+LastExceptionClassName+'
Exception Message: '+LastExceptionMessage);
  }
 }
```

10.10. Diğer Bileşen Tipleri

Konu kapsamında, bileşen grupları ile neler yapılabileceği ve kullanımları hakkında bir bilgiye yer verildi. Tüm bu bileşen gruplarının haricinde dâhili ve harici bileşenler bu bölümde ele alınmıştır.

10.10.1. TclStringList

TclStringList, bir dizi diziyi depolamak ve yönetmek için kullanılan, liste benzeri bir yapı sağlayan kullanışlı bir sınıftır. *TclStringList*, listedeki dize öğelerinde ekleme, kaldırma, sıralama, arama ve diğer işlemleri gerçekleştirmektedir.

TclStringList tipinde tanımlanan değişkeni kullanabilmek için nesneyi *StringListNew* fonksiyonu ile oluşturmak gerekmektedir.

Aşağıdaki örnekte; liste oluşturma, listeye veri ekleme ve mesaj kutusuyla gösterme işlemi yapılmıştır.

```
var
  List: TclStringList;
  i :Integer;

{
  List = Clomosy.StringListNew;

  try
    List.Add('Öğe 1');
    List.Add('Öğe 2');
    List.Add('Öğe 3');
    List.Add('Öğe 4');
    List.Add('Öğe 5');
    List.Add('Öğe 6');
    List.Add('Öğe 7');

    for (i = 0 to List.Count - 1)
      ShowMessage(Clomosy.StringListItemString(List,i));
  finally
    List.Free;
  }
}
```

Ekleme ve listeleme dışında silme, parçalama gibi farklı işlemler de yapılmaktadır. Aşağıda özellikler gösterilmiştir. Ayrıntılı bilgi almak isterseniz Clomosy doküman sitesine (docs.clomosy.com) bakabilirsiniz.

Özellik Kullanımı	Amacı
List.**Capacity** = 5;	Liste için ayrılan bellek alanının boyutunu temsil eder.
ItemCount = MyList.**Count**;	Listedeki öğelerin sayısını döndürür.
List.**Sorted** = True;	Listedeki öğeleri sıralama işlemi yapar. True değeri verildiği anda sıralama işlemi gerçekleşir.
List.**Insert**(1, 'Grafik');	Belirli bir konuma yeni bir öğe eklemek için kullanılır. Örnekte 1.dizine Grafik dizesini ekleme işlemi yapar.
List.**Delete**(1);	Belirli bir dizindeki öğeyi listeden silmek için kullanılır.
List.**Clear**;	Listedeki tüm öğeleri temizlemek için kullanılır.
Index = List.**IndexOf**('Clomosy');	Listedeki belirli bir öğenin dizinini bulmak için kullanılır.
List.**Exchange**(0,2);	Listedeki iki öğenin yerini değiştirmek için kullanılır.
List.**Sort**;	Listedeki öğeleri alfabetik olarak sıralamak için kullanılır.

10.10.2. TclHttp

Ağ üzerinden veri alışverişi yapmak, web servisleri ile iletişim kurmak veya internet üzerinden veri gönderip almak gibi işlemler için kullanılmaktadır. TclHttp, Clomosy'nin internet sayfaları ve web tabanlı uygulamalar oluşturmak için kullanılan bir dizi bileşeninden biridir. Bu bileşen HTTP üzerinden veri göndermeyi ve almayı yönetmektedir.

Bileşeni oluşturmak için *Create* fonksiyonu, bir URL adresine 'get' isteği göndermek için *GetRequest* fonksiyonu kullanılmaktadır.

```
var
  Str:String;
  MyHttp:TclHttp;

{

  MyHttp=TclHttp.Create(Nil);

  Try
   str=MyHttp.GetRequest('http://ipinfo.io/json');
   ShowMessage(str);
  Finally
   MyHttp.Free;
  }
}
```

10.10.3. TclRest

Clomosy'de *TclRest* bileşeni, veri aktarımı için RESTful servisleri ile etkileşimi kolaylaştıran bir bileşendir. REST (Temsili Durum Transferi), web tabanlı hizmetler arasında veri aktarımı için popüler bir mimaridir. Clomosy'deki *TclRest* bileşeni sayesinde bu tür servislerle kolayca iletişim kurulabilmekte, veri alışverişi yapabilmektedir.

TclRest bileşenini kullanarak bir RESTful servisine (GET, POST vb.) istekte bulunulur, yanıt alınır ve bu yanıtlar işlenebilmektedir.

Bileşeni oluşturmak için *Create* fonksiyonu kullanılır.

Diğer özellikleri aşağıdaki tabloda yer almaktadır.

Özellik Kullanımı	Amacı
Rest1.**BaseURL** = 'https://dummyjson.com/auth/login';	Erişilecek URL adresi verilir.
Rest1.**Accept** = 'application/json';	Sunucunun yanıtta kabul edeceği ortam türünü belirtir. Örnekte verilerin JSON formatında alınacağı belirtilmiştir. Accept özelliği ayrıca varsayılan olarak değerleri JSON biçiminde gönderir.
Rest1.**Method** = rmPOST; //rmGET	Yapılacak HTTP isteğinin türünü belirtir. Bir kaynağa yeni veri eklemek veya mevcut olanı POST isteğiyle güncellemek isteniyorsa *rmPOST* kullanılır. Bir kaynağın verilerini GET isteğiyle almak isteniyorsa *rmGET* kullanılmaktadır.
Rest1.**AddBody**('{"kullanıcı_adı":"kminchelle", "password":"0lelplR"}','application/json');	Sağlanan verileri belirtilen isteğin gövdesine ekler. Bu gövde POST gibi isteklerde verinin gönderildiği kısmı oluşturur. İstekte belirtilen veriler bu gövde aracılığıyla sunucuya iletilir. İki parametre alır. İlk parametre POST istek gövdesinde gönderilecek verileri belirtir, ikinci parametre ise gönderilen verilerin ortam türünü belirtir.
Rest1.**AddHeader**('Authorization','Bearer eyJhbGciO9.eyJpZCI6M');	Belirtilen başlığı HTTP isteğine ekler. İki parametre alır: ilk parametre başlık adıdır ve ikinci parametre sunucuya gönderilen verinin medya türü hakkında bilgi verir. Örnekte, "Authorization" başlığı kimlik doğrulama bilgilerini taşımak için kullanıldı. İkinci parametredeki 'Authorization' başlığının değeri bir JWT'yi (JSON Web Token) temsil eder. Bu belirteç, kullanıcının kimliğinin

	belirlenmesi için sunucu tarafından doğrulanan, doğrulanmış ve güvenli bir veri parçasıdır.
Rest1.**Execute**;	Bu yöntem çalıştırıldığında *TclRest* bileşeni belirtilen isteği gerçekleştirir ve sunucudan yanıtı alır.
Rest1.**Response**	Yanıtın detaylarına ve içeriğine sunucudan ulaşılır.

Aşağıdaki örnekte, basit bir kullanıcı arayüzü ile RESTful API'ye bir POST isteği gönderme gösterilmektedir. Ana blok içinde, form oluşturulduğunda, *TclRest* bileşeni ile belirtilen API URL'sine bir POST isteği gönderilir ve sunucudan gelen yanıt bir mesaj kutusunda kullanıcıya gösterilir. Örnekte, API'nin temel URL'si, gönderilecek verinin JSON formatında olduğu ve medya tipinin "application/json" olduğu belirtilmiştir. Ayrıca, bir kimlik doğrulama token'ı ile API'ye yetkilendirme bilgileri de eklenmişti, ancak bu kısım yorum satırı haline getirilmiştir. Bu örnek, Clomosy ile RESTful API isteklerini nasıl göndereceğinizi anlamak için iyi bir başlangıç noktasıdır.

```
var
  Form1 : TclForm;
  clRest : TCLRest;
  Button1 : TCLButton;

 void GetPostMethod;
 {
  clRest.Execute;
  ShowMessage(clRest.Response);
 }

 {
  Form1 = TclForm.Create(Self);
  Button1 = Form1.AddNewButton(Form1,'Button1', 'Click');
  Form1.AddNewEvent(Button1, tbeOnClick, 'GetPostMethod');

  clRest=TCLRest.Create;
  // Post
  clRest.BaseURL = 'https://dummyjson.com/auth/login';
  clRest.Accept = 'application/json';
  clRest.Method = rmPOST;
  clRest.AddBody('{"username":"kminchelle","password":"0lelplR"}','application/json');
  //clRest.AddHeader('Content-Type','application/json');

  /*
  // Get
  clRest.BaseURL = 'https://dummyjson.com/auth/me';
  clRest.Accept = 'application/json';
  clRest.Method = rmGET;
```

```
  clRest.AddHeader('Authorization','Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MTUsInVzZXJuYW1lIjoia21pbmNo
ZWxsZSIsImVtYWlsIjoia21pbmNoZWxsZUBxcS5jb20iLCJmaXJzdE5hbWUiOiJKZWFu
bmUiLCJsYXN0TmFtZSI6IkhhbHZc);
  */
  Form1.Run;
 }
```

10.10.4. TclTimer

TclTimer, belirli bir zaman aralığında, belirli bir işlemi gerçekleştirmek ve genellikle zaman tabanlı olayları veya periyodik görevleri uygulamak amacıyla kullanılmaktadır. *AddNewTimer* fonksiyonu kullanılarak oluşturulur ve üç parametre almaktadır. İlk parametre, tanımlanacak olan nesnenin hangi bileşen içerisinde konumlanacağını belirler. İkinci parametre, tanımlanan nesnenin ismini temsil eder. Üçüncü parametre ise milisaniye cinsinden çalışma zamanının ayarlanmasını ifade etmektedir.

Ayarlanan zaman diliminde, istenilen komutların çalışması için 'tbeOnTimer' adında olay tetikleyici kullanılmaktadır.

Zamanlayıcı bileşeninin çalıştırılacağı aralıkları belirtmek için *Interval* özelliği kullanılmaktadır. Bu özelliğe milisaniye cinsinden Integer bir değer ataması yapılmaktadır.

Bu tanımlamaları yaptıktan sonra zamanlayıcının aktif olmasını sağlamak için *Enabled* özelliği true (doğru) olmalıdır. Zamanlayıcı çalıştırıldıktan sonra durdurulmak istenirse bu sefer false (yanlış) olarak ayarlanmalıdır.

Aşağıdaki örnekte her bir saniyede 10'dan geriye doğru saydırma işlemi yapılmıştır. Sayaç 0 olduğu anda mesaj kutusunda Süre doldu mesajı verilmektedir.

```
var
  anaForm:TclForm;
  GetTimer: TClTimer;
  sayac : Integer;
  lblTimer : TclLabel;

void zamanlayiciBaslat;
{
 if (sayac == 0)
 {
   GetTimer.Enabled = False;
   lblTimer.Caption = IntToStr(sayac);
   ShowMessage('Süre doldu!');
 }
 else
 {
   lblTimer.Caption = IntToStr(sayac);
```

```
    Dec(sayac);
  }
}

{
  sayac = 10;
  anaForm = TclForm.Create(Self);

  GetTimer = anaForm.AddNewTimer(anaForm,'GetTimer',1000);
  GetTimer.Enabled = True;
  anaForm.AddNewEvent(GetTimer,tbeOnTimer,'zamanlayiciBaslat');

  lblTimer= anaForm.AddNewLabel(anaForm,'lblTimer','');
  lblTimer.StyledSettings = ssFamily;
  lblTimer.TextSettings.Font.Size=35;
  lblTimer.Align = alCenter;
  lblTimer.Height = 50;
  lblTimer.Width = 50;
  anaForm.Run;
}
```

10.11. Bileşenlerin Ortak Özellikleri

Kullanıcı arayüzünde kullanılan bileşenlerin çoğu temelde aynı sınıftan oluştuğu için doğal olarak birçok özellikleri de ortaktır. Bu ortak özelliklerin öğrenilmesi durumunda, hâlihazırda yer alan bileşenlerin veya sonradan kurulacak birçok bileşenin kullanımı daha kolay olacaktır.

Standart bileşenler ve profesyonel bileşenlerin farklı kullanımları bulunmaktadır. Profesyonel bileşenlerde özellik tanımlamaları yapabilmek için clProSettings (önerilen) ve SetupComponent (JSON yapısıyla) yapısına başvurulmaktadır.

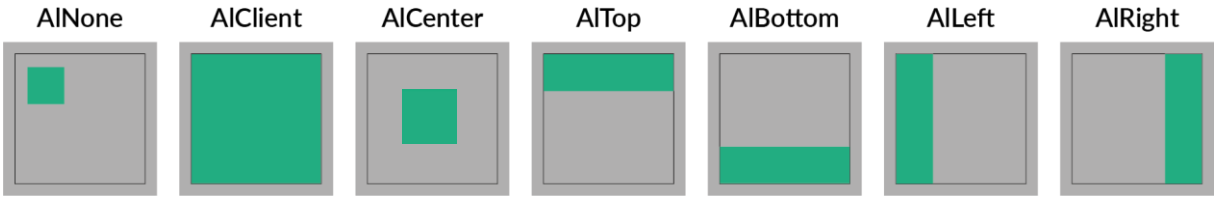
10.11.1. Bileşen Hizalama

Tüm bileşenlerin farklı cihaz çözünürlüklerine aynı davranışı göstermesi beklenemez. Örneğin, çalışma anında cihaz çözünürlüğüne veya pozisyonuna göre bileşenler ekran sınırlarını aşabilir veya yerleştiği konumdan daha farklı bir konumda görünebilir. Bu durumda bileşenin konumunu sabitlemeye yönelik bir takım zorlayıcı özelliklerin kullanılması gerekmektedir.

Align

Nesnelerin bulunduğu konumda bir veya daha fazla tarafa hizalanması için bileşenlerin temel sınıfına ait *Align* özelliği kullanılmaktadır. Bu tipler form veya benzeri bir taşıyıcı nesne üzerinde bileşenleri hizalamaktadır. Nesneye bu özellik atadığında üzerinde bulunduğu kontrolün içinde sağa, sola, üste, alta, merkeze veya tümüne otomatik olarak hizalanmaktadır. Align özelliğinin varsayılan değeri **None** olarak atanmıştır. None özelliğine sahip bir nesne yerleştirildiği noktada sabit kalır.

Align özelliğinin en yaygın kullanılan tipleri aşağıda verilmiştir.



Sık kullanılan diğer Align tipleri aşağıda daha detaylı olarak anlatılmıştır.

AlContents: Client tipinde olduğu gibi, üzerine yerleşik olduğu nesnenin sınırlarını kaplayacak şekilde yerleşir. Client tipinden farklı olarak formun tamamını kapsamaktadır.

AlFit: Ana alana sığacak şekilde, nesnenin en oranı korunarak merkeze konumlandırılır.

AlFifLeft: Ana alana sığacak şekilde, nesnenin en oranı korunarak sola konumlandırılır.

AlFifRight: Ana alana sığacak şekilde, nesnenin en oranı korunarak sağa konumlandırılır.

AlHorizontal: Nesnenin yüksekliği etkilenmeden bulunduğu yerde yatay konumda sağ ve sol sınırları doldurularak hizalanır.

AlHorzCenter: Nesnenin genişliği etkilenmeden dikey konumda alt ve üst sınırları doldurularak merkeze hizalanır.

AlMostBottom: Nesne yüksekliği etkilenmeden en alta konumlandırılır. Daha önce AlBottom ile konumlandırılan nesneler var ise bunların da altına gelecek şekilde yerleştirilir.

AlMostLeft: Nesne genişliği etkilenmeden en sola konumlandırılır. Daha önce AlLeft ile konumlandırılan nesneler var ise bunların da soluna gelecek şekilde yerleştirilir.

AlMostRight: Nesne genişliği etkilenmeden en sağa konumlandırılır. Daha önce AlRight ile konumlandırılan nesneler var ise bunların da sağına gelecek şekilde yerleştirilir.

AlMostTop: Nesne yüksekliği etkilenmeden en üste konumlandırılır. Daha önce AlTop ile konumlandırılan nesneler var ise bunların da üstüne gelecek şekilde yerleştirilir.

Scale: Üzerinde bulunduğu kontrolün yükseklik ve genişliği yeniden boyutlandırıldığında; nesnenin yükseklik ve genişliği otomatik olarak ayarlanır. Yatay ve dikey konumu da yeniden boyutlandırmaya bağlı olarak, orantılı bir şekilde değişir.

AlVertical: Nesnenin genişliği etkilenmeden bulunduğu yerde dikey konumda alt ve üst sınırları doldurularak hizalanır.

AlVertCenter: Nesnenin yüksekliği etkilenmeden yatay konumda sağ ve sol sınırları doldurularak merkeze hizalanır.

Örnek kullanımı:

```
Button1.Align = AlTop;
```

Locked

Tasarım anında nesnenin yerleştirildiği konumda sabit kalmasını sağlamaktadır. Varsayılan değeri false'dur, true olması durumunda form üzerine yerleştirilen nesnenin yeri değiştirilmez.

```
Button1. Locked = True;
```

Margins

İç içe yerleştirilmiş iki nesneden, içteki nesnenin dıştaki nesneye uyguladığı mesafeyi ifade etmektedir.

İçerisinde bulunduğu nesnenin kendi kenarlarına olan uzaklığını belirlemek için kullanılmaktadır. İki nesne arasındaki mesafe piksel cinsindendir. *Margins* özelliği yalnızca *Align* ile hizalanmış nesnelere uygulanabilir. Nesne üzerinde kullanılan özellikler aşağıdaki gibidir:

Bottom: Bir nesnenin, üzerinde bulunduğu nesnenin alt tarafına uyguladığı mesafeyi ifade etmektedir.

Left: Nesnenin sol tarafına uyguladığı mesafeyi ifade etmektedir.

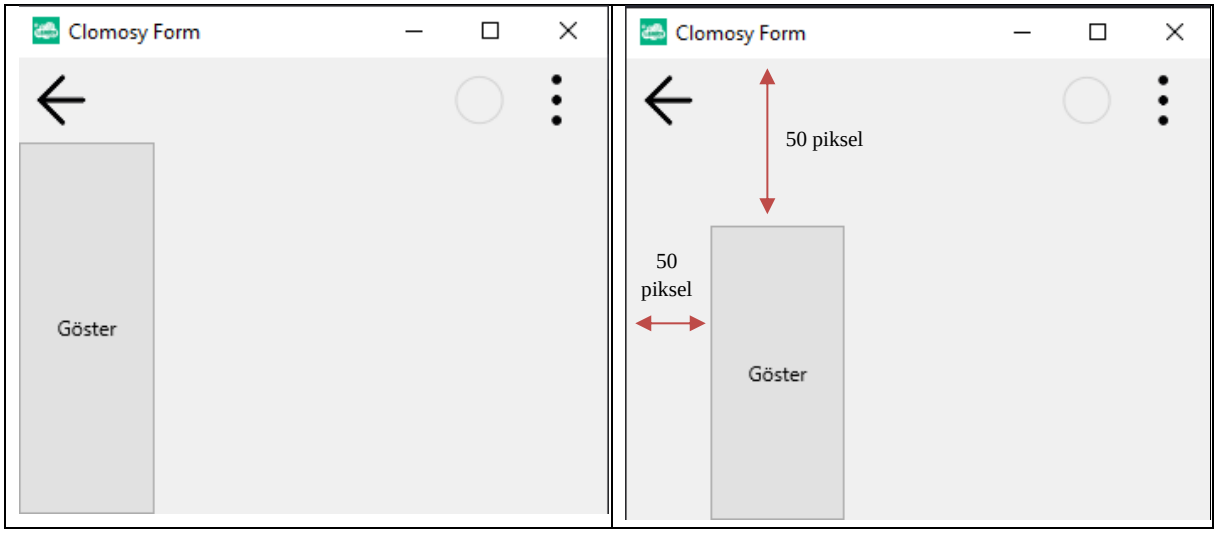
Right: Nesnenin sağ tarafına uyguladığı mesafeyi ifade etmektedir.

Top: Nesnenin üst tarafına uyguladığı mesafeyi ifade etmektedir.

Örnek kullanımı:

```
Button1.Align = AlLeft;
Button1.Margins.Top = 50;
Button1.Margins.Left = 50;
```

Yukarıdaki kod yapısında; Button1, öncelikle üzerinde bulunduğu nesnenin sol tarafına hizalanmakta, üst ve sol kenarında 50 piksel boşluk olacak şekilde konumlandırılmaktadır.



Resim 60

Padding

Margins özelliğinde olduğu gibi iç içe yerleştirilen iki nesne arasındaki uzaklığı belirlemek için kullanılmaktadır. *Margins* özelliğinde içteki nesne mesafeyi belirlerken, *padding* özelliğinde ise dıştaki nesne, kenarlara olan uzaklığı belirlemektedir. İki nesne arasındaki mesafe piksel cinsindendir.

Padding özelliği sadece *Align* ile hizalanmış nesnelere uygulanmaktadır. Atanabilen tipleri aşağıdaki gibidir.

Bottom: Bir nesnenin, kendi içerisinde bulunan diğer bir nesnenin alt tarafına uyguladığı mesafeyi ifade etmektedir.

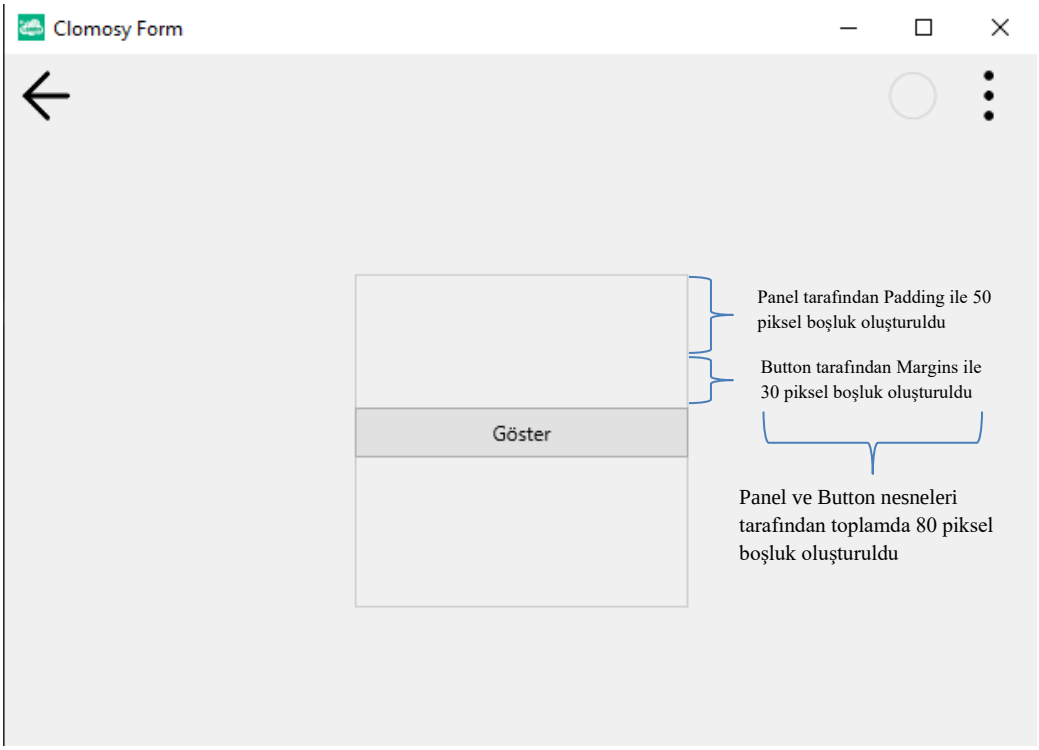
Left: İç kısımda bulunan nesnenin sol tarafına uyguladığı mesafeyi ifade etmektedir.

Right: İç kısımda bulunan nesnenin sağ tarafına uyguladığı mesafeyi ifade etmektedir.

Top: İç kısımda bulunan nesnenin üst tarafına uyguladığı mesafeyi ifade etmektedir.

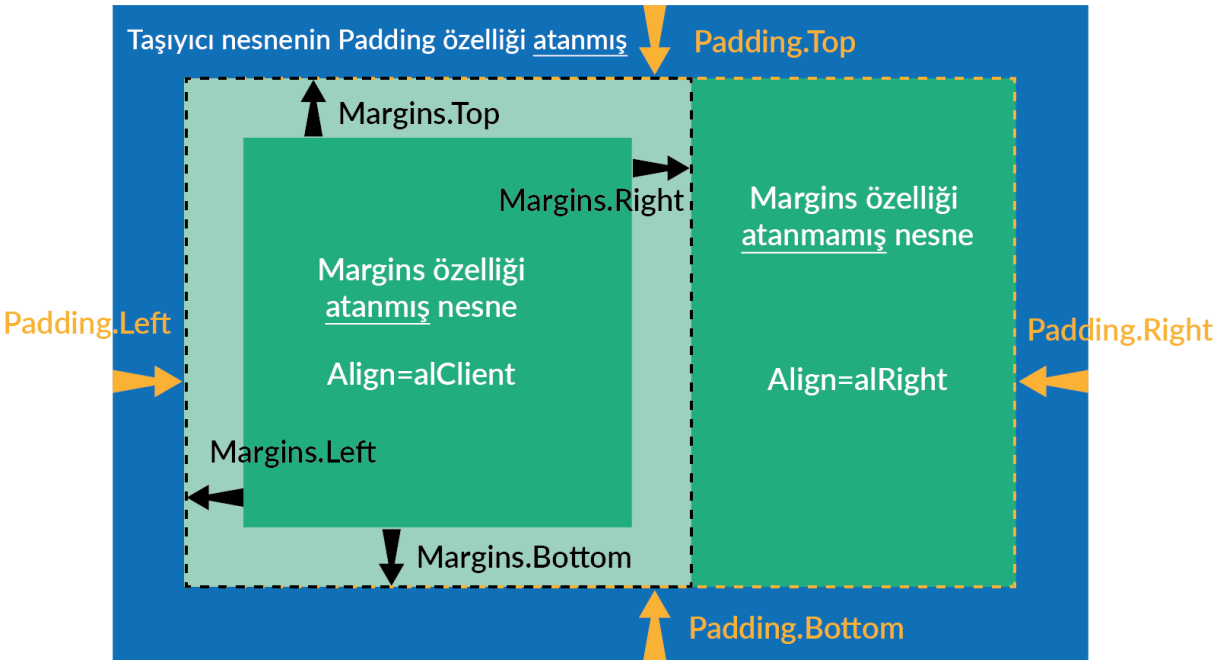
Padding ve margins arasındaki mesafeyi daha iyi anlamak için aşağıdaki örneği inceleyelim.

```
Panel1.Padding.Top = 50;
Button1.Align = AlTop;
Button1.Margins.Top = 30;
```



Resim 61

Aşağıdaki ise Align, Padding ve Margins özelliklerinin birbirlerine etkisi gösterilmektedir.



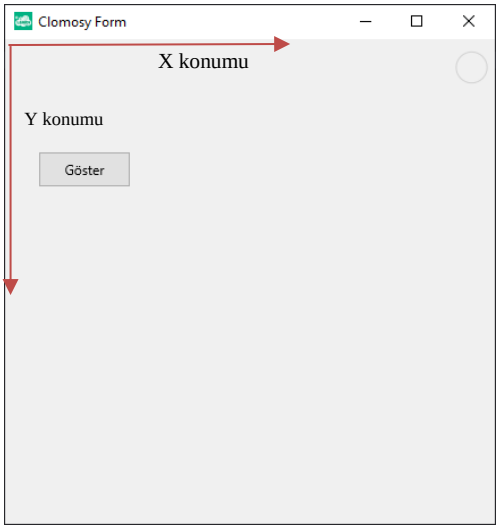
Resim 62

Position

Form içerisinde bir nesnenin konumunu belirtmek için kullanılmaktadır. X, nesnenin yataydaki konumunu, Y ise dikeydeki konumunu ifade etmektedir.

Örnek kullanımı

```
Button1.Position.X = 30;
Button1.Position.Y = 100;
```



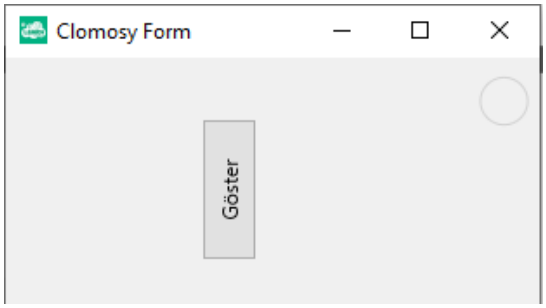
Resim 63

RotationAngle

Nesnenin X eksenindeki dönüş açısını göstermektedir. Bu değer derece cinsindendir. Negatif veya pozitif değer alabilir. RotationAngle pozitif olması durumunda saat yönünde negatif olması durumunda ise tersi yönde döndürülmektedir.

```
Button1.RotationAngle = -90;
```

Yukarıdaki kod parçasının görünümü aşağıda gösterilmektedir.



Resim 64

RotationCenter

Nesnenin X ve Y ekseni üzerinde dönüş merkezini belirtmektedir. X ve Y, 0 ile 1 arasında değer alır. X=0 ve Y=0 olması durumunda sol üst köşe, X=1 ve Y=1 olması durumunda ise sağ alt köşe dönüş merkezi olacaktır.

10.11.2. Bileşen Boyutlandırma

Daha çok basit ve sade arayüz tasarımları veya küçük çözünürlüklere sahip cihazlarda bileşen boyutlandırma önem taşımaktadır. Örneğin 10 inch'lik bir çözünürlüğe sahip cihaz ile 12 inch'e sahip cihaz arayüzünde kullanılan nesne boyutlarının aynı olması, uygulamanın kullanılabilirliği zorlaştıracaktır. Bu anlamda bileşenlerin yeniden boyutlandırılması önem arz eder.

Height

Görsel nesnelerin yüksekliğini belirler. Height değeri piksel cinsindendir.

Örnek kullanımı:

```
Panel1.Height = 80;
```

Width

Görsel nesnelerin genişliğini belirtir. Width değeri piksel cinsindendir.

Örnek kullanımı:

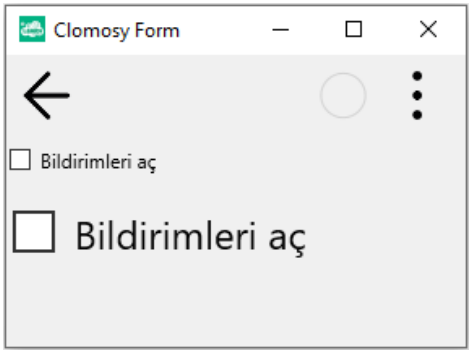
```
Panel1.Width = 100;
```

Scale

Nesnenin X ve Y ekseni üzerinde ölçeğini belirtmektedir. X ve Y için varsayılan değer 1'dir. Değerlerin 2 olması durumunda bileşen boyutu iki kat artacaktır.

Aşağıdaki şekilde form üzerine 2 adet TclCheckBox nesnesi yerleştirilmiştir. İkinci bileşenin *Scale* değerleri, X=2 ve Y=2 atandığında nesne boyutunun da 2 kat arttığı görülmektedir.

```
CheckBox1.Scale.X = 2;
CheckBox2.Scale.Y = 2;
```



Resim 65

10.11.3. Bileşen Görünürlüğü

Bileşenlerin tasarım anı veya çalışma anında görünürlüğünü belirleyen özelliklerdir.

Visible

Tasarım ve çalışma anında nesnenin görünürlük durumunu belirtmektedir. Varsayılan değeri true'dur. False olması durumunda nesne görünmez olur.

```
CheckBox1.Visible = False;
```

10.11.4. Diğer Temel Bileşen Özellikleri

Enabled

Çalışma anında, nesne veya nesnelerin bulunduğu kontrol grubunun aktif veya pasif olmasını belirtmektedir. Varsayılan değeri true'dir. False olması durumunda uygulama çalışma anında nesne pasif olacak ve son kullanıcı tarafından müdahale edilmesine izin verilmeyecektir.

```
Button1.Enabled = False;
```

EnableDragHighlight

Çalışma anında, nesne üzerine başka bir nesne sürüklendiğinde, kenar kısmında ışıklı bölüm oluşturarak vurgulanması sağlanmaktadır. Varsayılan değeri true'dur. False olması durumunda sürüklenen nesne, üzerine geldiğinde vurgu oluşmaz.

```
Button1.EnableDragHighlight = False;
```

HitTest

Bir bileşenin dokunma veya tıklanma olayını yakalamakta ve bu olayın çalışıp çalışmayacağını belirlemektedir. **True** olması durumunda, nesnenin *tbeOnClick* olayı çalışır, **False** durumunda ise çalışmaz. Varsayılan değeri **True**'dur. Fakat *TclLabel* nesnesinde varsayılan değer **false**'dur.

Bir nesnenin *HitTest* özelliği **False** ise kullanıcı tarafından bu nesneye tıklanırsa, üzerinde bulunduğu ana nesnenin tbe*OnClick* olayı çalışacaktır. Aksi durumunda nesnenin kendi tbe*OnClick* özelliği çalışacaktır.

Örnek verecek olursak forma bir panel ve panelin içine de bir buton yerleştirelim. Panel ve buton nesnelerine *tbeOnClick* olayı eklenerek aşağıdaki kodları yazalım ve uygulamayı çalıştıralım.

```
void PanelClicked;
{
  ShowMessage('Panele tıklandı.');
}

void ButtonClicked;
{
  ShowMessage('Butona tıklandı.');
}
```

Panel ve buton nesnelerine ayrı ayrı tıklandığında ikisi de kendi *tbeOnClick* olayı çalışacaktır. Bunun ardından butonun *HitTest* özelliğini false yapalım ve çalıştıralım.

```
Button1.HitTest = False;
```

Programda panel ya da butona tıklandığında panelin *tbeOnClick* olayı tetiklenecek ve Panele tıklandı mesajı görüntülenecektir.

Opacity

Nesnelerin saydamlık durumunu belirtmektedir. Opacity, 1 ile 0 arasında bir değer alır. Varsayılan değeri 1'dir. Opacity değeri 0 olursa nesne tasarım ve çalışma anında tamamen şeffaf olur.

Opacity ve visible özellikleri arasındaki en önemli fark; çalışma anında bir nesnenin Visible değeri false olduğunda nesneye ait tbeOnClick gibi kullanıcı etkileşimli olaylar çalışmaz. Opacity değeri sıfır olan bir nesne de çalışma anında görünmez fakat tbeOnClick özelliği kullanılabilir.

```
Button1.Opacity = 0.25;
```

Hint

Masaüstü uygulamalarda kullanıcının mouse ile bir bileşen üzerine geldiğinde gösterilen küçük ipuçlarıdır. İpucu için Hint özelliğine bir değer içermesi gerekmektedir.

```
Button1.Hint = 'Button1 Text';
```

Tag

Her nesne için ayrı bir tamsayı değeri barındırmaktadır. Varsayılan değeri sıfır (0)'dır. Çalışma veya tasarım anında bu değer etiket olarak kullanılmaktadır.

```
Button1.Tag = 12;
```

SetFocus

Bir nesnenin odak (focus) almasını sağlayan bir yöntemdir. Odak, genellikle bir kullanıcının bir formda, bir kontrolde (örneğin, bir edit kutusunda veya bir düğmede) bulunduğu alanı belirtmektedir. Özellikle, bir form yüklendiğinde veya belirli bir olay gerçekleştiğinde bir kontrolün otomatik olarak odak almasını sağlamak için kullanılmaktadır. Bu, kullanıcının manuel olarak bir kontrolü seçmesine gerek kalmadan, örneğin bir edit kutusuna doğrudan yazmaya başlamak gibi durumları ele alabilir.

```
Edit1.SetFocus;
```

Text

Text özelliğini kullanarak, genellikle kullanıcının girdiği metni veya bileşenin içinde bulunan metni elde edebilir veya ayarlanabilir. Text özelliği, genellikle metin tabanlı bileşenlerde (örneğin, TclEdit, TclMemo, TclLabel gibi) bulunur.

Örnek olarak TclEdit bileşeninin içindeki metni almak için;

```
var
  metin: string;
{
  metin = Edit1.Text;
}
```

Edit1 adlı TclEdit bileşenine metin atamak için;

```
Edit1.Text = 'Yeni Metin';
```

Caption
Caption özelliği, özellikle görsel bileşenlerde (visual components) bulunan ve genellikle kullanıcı arayüzü elemanlarının üzerinde görünen metni temsil eder. Bu özellik, metni görsel olarak temsil eden bileşenler için kullanılır ve genellikle etiketler, butonlar, formlar gibi bileşenlerde bulunur.

```
Button1.Caption = 'Ana Button';
```

ClTypeOfField
Bir metin girdileri bileşenlerinde istenilen her karakter yazılabilmektedir. Karakter sınırlaması getirmek istendiğinde bu özellik kullanılır. Özelliğe taFloat ataması yapıldığında yalnızca sayılar ve virgül karakterleri yazılabilir. taString ataması yapıldığında ise tüm karakter yazılabilir (sayılar, semboller, harfler).

```
Edit1.clTypeOfField = taFloat;
```

ReadOnly
Bu özellik, bir bileşenin içeriğini kullanıcının düzenlemesine izin verip vermemeyi kontrol eder. Eğer ReadOnly özelliği True olarak ayarlanırsa, kullanıcı bileşenin içeriğini değiştiremez; ancak, False olarak ayarlanırsa, kullanıcı içeriği düzenleyebilir.

```
Edit1.ReadOnly = True;
```

MaxLength
MaxLength özelliği, bir metin girişi alanındaki (genellikle TclEdit veya TclMemo gibi bileşenlerde) maksimum karakter sayısını belirten bir özelliktir. Bu özellik, kullanıcının bir metin giriş alanına belirli bir uzunluktan fazla karakter girmesini engellemek için kullanılır.

```
Edit1.MaxLength = 2;
```

TextSettings
Standart bileşen kullanımlarında bir metne boyutlandırma ve renk verme gibi özellikleri kullanabilmek için TextSettings özelliği kullanılır. Bu bileşenin (bir form, bir düğme veya başka bir bileşen) stillendirme ayarlarını belirler. Bu özelliğin birden fazla kullanımı mevcuttur.

```
Button1.StyledSettings = ssFamily;
```

ssFamily değeri, bileşenin aile temelli stil ayarlarını kullanmasını ifade eder. Bu, bileşenin ailesi tarafından tanımlanan stil özelliklerini kullanmasını sağlar.

StyledSettings özelliği ayarlaması yapıldıktan sonra metnin yazı boyutu *Font.Size* özelliği ile belirlenebilir.

```
Button1.TextSettings.Font.Size = 50;
```

Bileşenin metin rengini ayarlamak için ise *FontColor* özelliği kullanılır.

```
Button1.TextSettings.FontColor = clAlphaColor.clHexToColor('#8a067c');
```

Ya da

```
Button1.TextSettings.FontColor = clAlphaColor.clblue;
```

Renk sabitlerini öğrenebilmek için https://www.docs.clomosy.com/Color_Constants sayfasını inceleyiniz.

KeyboardType

KeyboardType, mobil uygulama geliştirmede sanal klavyenin türünü belirtmek için kullanılan bir özelliktir. Bu, kullanıcılar belirli bir alana veri girdiğinde uygun klavye tipini otomatik olarak açarak kullanıcı deneyimini geliştirir.

Klavye Türleri

KeyboardType, kullanıcının gireceği veri türüne bağlı olarak çeşitli klavye türlerini içerir. KeyboardType değerleri şunlardır:

Tür	Kullanım Amacı
vktDefault	Varsayılan klavye türü
vktNumsAndPunctuation	Sayılar ve noktalama işaretleri için klavye
vktNumberPad	Yalnızca sayılar için sayısal tuş takımı
vktPhonePad	Telefon numarası girişi için klavye
vktAlphabet	Alfabetik karakterler için klavye
vktURL	URL girişi için klavye
vktEmailAddress	E-posta adresi girişi için klavye
vktNamePhonePad	Sayılar ve alfabetik karakterler için klavye
vktDecimalNumberPad	Ondalık sayılar için sayısal klavye

Kullanımı:

```
Edit1.KeyboardType = vktPhonePad;
```

Aşağıdaki örnekte üç farklı edit üzerinde kullanıcı iletişim bilgileri alınmaya çalışılmaktadır. *KeyboardType* özelliği kullanıcı adı ve soyadı için *vktDefault*, e-posta için *vktEmailAddress* ve telefon numarası bilgisi için ise *vktPhonePad* olarak ayarlanmaktadır.

```
 var
  anaForm:TclForm;
  pnlAdSoyad,pnlEPosta,pnlTel:TclProPanel;
  lblBaslik, lblAdSoyad,lblEPosta,lblTel:TClLabel;
  edtAdSoyad,edtEPosta,edtTel:TclEdit;

{
  anaForm=TclForm.Create(Self);

  lblBaslik=anaForm.AddNewLabel(anaForm,'lblBaslik','İLETİŞİM BİLGİLERİ');
  lblBaslik.Align=alMostTop;
  lblBaslik.StyledSettings = ssFamily;
  lblBaslik.TextSettings.Font.Size = 20;
  lblBaslik.TextSettings.FontColor = clAlphaColor.clHexToColor('#8a067c');
  lblBaslik.Margins.Left= 10;
  lblBaslik.Margins.Top= 10;
  lblBaslik.Height = 50;

  pnlAdSoyad=anaForm.AddNewProPanel(anaForm, 'pnlAdSoyad');
  pnlAdSoyad.Align=alTop;
  pnlAdSoyad.Height = 100;

  lblAdSoyad=anaForm.AddNewLabel(pnlAdSoyad,'lblAdSoyad','Ad/Soyad');
  lblAdSoyad.Align=alLeft;
  lblAdSoyad.Width = pnlAdSoyad.Width/2-20;
  lblAdSoyad.TextSettings.Font.Size = 20;

  edtAdSoyad = anaForm.AddNewEdit(pnlAdSoyad,'edtAdSoyad','Ad ve soyad giriniz...');
  edtAdSoyad.Align = alClient;
  edtAdSoyad.StyledSettings = ssFamily;
  edtAdSoyad.KeyboardType = vktDefault;


  pnlEPosta=anaForm.AddNewProPanel(anaForm, 'pnlEPosta');
  pnlEPosta.Align=alTop;
  pnlEPosta.Height = 100;

  lblEPosta=anaForm.AddNewLabel(pnlEPosta,'lblEPosta','E-Posta');
  lblEPosta.Align=alLeft;
  lblEPosta.Width = pnlEPosta.Width/2-20;
  lblEPosta.TextSettings.Font.Size = 20;

  edtEPosta = anaForm.AddNewEdit(pnlEPosta,'edtEPosta','E-posta giriniz...');
  edtEPosta.Align = alClient;
  edtEPosta.StyledSettings = ssFamily;
  edtEPosta.KeyboardType = vktEmailAddress;

  pnlTel=anaForm.AddNewProPanel(anaForm, 'pnlTel');
  pnlTel.Align=alTop;
```

```
  pnlTel.Height = 100;

  lblTel=anaForm.AddNewLabel(pnlTel,'lblTel','Tel');
  lblTel.Align=alLeft;
  lblTel.Width = pnlTel.Width/2-20;
  lblTel.TextSettings.Font.Size = 20;

  edtTel = anaForm.AddNewEdit(pnlTel,'edtTel','Telefon numaranızı giriniz...');
  edtTel.Align = alClient;
  edtTel.StyledSettings = ssFamily;
  edtTel.KeyboardType = vktPhonePad;
  anaForm.Run;
}
```

BÖLÜM 11. Sensör Yapıları

Akıllı cihazlar üzerinde birçok sensör yer almaktadır. Sensörler son kullanıcıların işlerini kolaylaştırması açısından büyük önem taşımaktadır. Örneğin pusula özelliği ile yön tayini yapabilir, hız ve hareket sensörleri ile oyunlarda karakter veya nesnelerin kontrol edilmesini sağlayabilirsiniz.

Konu kapsamında, mobil platformlarda bulunan sensör grupları, sensörlerin nasıl kullanılacağı, ne tür veriler elde edileceği ve yazılım gelişim aşamasında neler yapılacağı örneklerle anlatılacaktır.

11.1. Pusula ve Denge Sensörü

Clomosy'de mobil uygulama geliştirme bağlamında cihazın yönünü tespit etmek için bir sensör kullanılır. *FormOrientationSensorType* özelliği, mobil formun cihazın yönünü izlemek için hangi sensör türünü kullanacağını belirler.

FormOrientationSensorType özelliği *ostTilt* olarak ayarlanmışsa denge eğimini izleyecektir. *ostHeading* olarak ayarlanmış ise cihazın pusula yani kuzeye doğru açısının belirlenmesi sağlar.

11.1.1. Pusula Sensörü

Yön tespit etmeye yarayan manyetik sensördür. Daha çok GPS, Navigasyon veya Dini uygulamalarda kullanılmaktadır. Cihazın hangi yöne yönlendirildiğini gösterir.

Yönlendirme sınıfının temel sınıfı ve *FormOrientationSensorType* enum tipi aşağıdaki özellikleri barındırmaktadır.

HeadingX, HeadingY, HeadingZ: Cihazın X, Y ve Z eksenleri üzerindeki dönme açılarını gösteren özelliklerdir.

```
//XHeading, YHeading, ZHeading : Double
XHeading = MyForm.FormOrientationSensor.Sensor.HeadingX;
YHeading = MyForm.FormOrientationSensor.Sensor.HeadingY;
ZHeading = MyForm.FormOrientationSensor.Sensor.HeadingZ;
```

Aşağıdaki örnekte bir pusula görselinin pusula sensörü ile hareketi gösterilmiştir.

```
var
  anaForm: TCLForm;
  hareketImg,yonImg: TClProImage;
  zamanlayici: TClTimer;

void sensorDurum;
var
  YHeading,XHeading,ZHeading: Double;
{
  if (not anaForm.FormOrientationSensor.Sensor == nil)
  {
    YHeading = anaForm.FormOrientationSensor.Sensor.HeadingY;
    XHeading = anaForm.FormOrientationSensor.Sensor.HeadingX;
    ZHeading = anaForm.FormOrientationSensor.Sensor.HeadingZ;

    hareketImg.RotationAngle = YHeading;
  }
}

{
  anaForm = TCLForm.Create(Self);
  anaForm.SetFormBGImage('https://clomosy.com/demos/compassbg.png');

  hareketImg = anaForm.AddNewProImage(anaForm,'hareketImg');
  hareketImg.Align = AlCenter;
  hareketImg.Width = 350;
  hareketImg.Height = 350;
  hareketImg.clProSettings.PictureSource = 'https://clomosy.com/demos/compass2.png';
  hareketImg.clProSettings.PictureAutoFit = True;
  hareketImg.SetclProSettings(hareketImg.clProSettings);

  yonImg = anaForm.AddNewProImage(anaForm,'yonImg');
  yonImg.Align = AlCenter;
  yonImg.Width = 350;
  yonImg.Height = 350;
  yonImg.Margins.Bottom = 100;
  yonImg.Margins.Right = 28;
  yonImg.clProSettings.PictureSource = 'https://clomosy.com/demos/compass_arrow2.png';
  yonImg.clProSettings.PictureAutoFit = True;
  yonImg.SetclProSettings(yonImg.clProSettings);
```

```
  anaForm.FormOrientationSensorType = ostHeading;
  anaForm.FormOrientationSensor.Active = True;

  zamanlayici = anaForm.AddNewTimer(anaForm, 'zamanlayici', 500);
  zamanlayici.Enabled = True;
  anaForm.AddNewEvent(zamanlayici, tbeOnTimer, 'sensorDurum');
  anaForm.Run;
}
```

11.1.2. Denge Sensörü

Denge sensörü terimi genellikle "ivmeölçer" (accelerometer) ya da "jiroskop" (gyroscope) gibi sensörleri ifade etmektedir. Bu sensörler, bir cihazın hareketini ve yönelimini ölçen, genellikle mikroelektronik cihazlarda ve mobil cihazlarda bulunan küçük sensörlerdir.

TiltX, TiltY, TiltZ: Cihazın X, Y ve Z konum sensörlerini gösteren özellikleridir.

```
 //YTilt, XTilt, ZTilt: Double
 XTilt = MyForm.FormOrientationSensor.Sensor.TiltX;
 YTilt = MyForm.FormOrientationSensor.Sensor.TiltY;
 ZTilt = MyForm.FormOrientationSensor.Sensor.TiltZ;
```

Aşağıdaki örnekte bir görseli denge sensörü ile X konumu ile yönlendirme işlemi yapılmaktadır.

```
var
  anaForm: TCLForm;
  dengeImg: TClProImage;
  zamanlayici: TClTimer;
void dengeDurum;
var
  XTilt: Double;

{

  try
    if (not anaForm.FormOrientationSensor.Sensor==nil)
    {
      XTilt = anaForm.FormOrientationSensor.Sensor.TiltX;
      dengeImg.RotationAngle = XTilt;
    }
  except
    ShowMessage('Exception Class: '+LastExceptionClassName+' Exception Message:
'+LastExceptionMessage);
  }
}
```

```
 {
 anaForm = TCLForm.Create(Self);
 anaForm.SetFormBGImage('https://clomosy.com/demos/compassbg.png');
 dengeImg = anaForm.AddNewProImage(anaForm,'dengeImg');
 dengeImg.Align = AlCenter;
 dengeImg.Width = 350;
 dengeImg.Height = 350;
 dengeImg.clProSettings.PictureSource = ' https://clomosy.com/demos/balance.png ';
 dengeImg.clProSettings.PictureAutoFit = True;
 dengeImg.SetclProSettings(dengeImg.clProSettings);

 anaForm.FormOrientationSensorType = ostTilt;
 anaForm.FormOrientationSensor.Active = True;
 zamanlayici = anaForm.AddNewTimer(anaForm, 'zamanlayici', 500);
 zamanlayici.Enabled = True;
 anaForm.AddNewEvent(zamanlayici, tbeOnTimer, 'dengeDurum');
 anaForm.Run;
}
```

11.2. Haraket Sensörü

Clomosy platformunda bulunan, cihazın hareket sensörünün değerlerini okumanızı sağlayan bir bileşendir. Hareket sensörü cihazın konumunu ve hızını algılar. Bu bileşen sayesinde cihazın fiziksel hareketleri takip etmektedir.

Hareket sensörü kullanımı için Clomosy yapısında *TClMotionSensor* sınıfına ait nesne oluşturulmalıdır. Bir sensör oluşturmak için *AddNewSensorsMotion* fonksiyonu kullanılır. Bu işlev iki parametre almaktadır. İlk parametre tanımlanacak olan sensörün hangi bileşen içerisinde konumlandırılacağını belirler. İkinci parametre tanımlanan sensör ismini temsil eder.

```
Var
   DeviceMotionSensor:TClMotionSensor;
{
 DeviceMotionSensor = anaForm.AddNewSensorsMotion(anaForm,'DeviceMotionSensor');
 DeviceMotionSensor.Active = True;
}
```

Cihaz x ekseninde hareket algıladığında döndürülen değer için *GetDirectionX*, cihaz y ekseninde hareket algıladığında, döndürülen değer için *GetDirectionY,* cihazın x ekseni boyunca ivmesini okumak için *SensorAccelerationX* özelliği kullanılmaktadır. Bu özellik x ekseni boyunca hızlanma, cihazın belirli bir yönde hızlandığını veya yavaşladığını göstermektedir. Aynı şekilde y ekseni için ise *SensorAccelerationY* özelliği kullanılır.

```
  ShowMessage(DeviceMotionSensor.GetDirectionX);
  ShowMessage(DeviceMotionSensor.GetDirectionX);
  ShowMessage(DeviceMotionSensor.SensorAccelerationX);
  ShowMessage(DeviceMotionSensor.SensorAccelerationY);
```

Bir örnek üzerinde bakalım. Örnekte, bir görsel x ve y konumuna göre hareket ettirildiğinde görselin pozisyonu değiştirilecektir.

```
Var
  anaForm:TclForm;
  DeviceMotionSensor:TClMotionSensor;
  zamanlayici:TClTimer;
  hareketSensorImg : TCLImage;

void sensorDurum;
Const
  BallSpeed = 5;
{
  If (Clomosy.PlatformIsMobile)
  {
    Case DeviceMotionSensor.GetDirectionX of
    {
      1:hareketSensorImg.Position.X = hareketSensorImg.Position.X - BallSpeed;
//Sola hareket et
      2:hareketSensorImg.Position.X = hareketSensorImg.Position.X + BallSpeed;
//Sağa hareket et
    }
    Case DeviceMotionSensor.GetDirectionY of
    {
      1:hareketSensorImg.Position.Y = hareketSensorImg.Position.Y - BallSpeed;//Yukarı
taşı
      2:hareketSensorImg.Position.Y = hareketSensorImg.Position.Y + BallSpeed;//Aşağı
taşı
    }
  }
}

{
  anaForm = TclForm.Create(Self);
  anaForm.SetFormBGImage('https://clomosy.com/demos/bg4.jpg');

  hareketSensorImg= anaForm.AddNewImage(anaForm,'hareketSensorImg');
  hareketSensorImg.Align = alCenter;
  hareketSensorImg.Align = alNone;
  hareketSensorImg.Height = 200;
  hareketSensorImg.Width = 200;
  anaForm.setImage(hareketSensorImg,'https://clomosy.com/demos/balloon2.png');
```

```
  DeviceMotionSensor =
anaForm.AddNewSensorsMotion(anaForm,'DeviceMotionSensor');
  DeviceMotionSensor.Active = True;

  zamanlayici= anaForm.AddNewTimer(anaForm,'zamanlayici',1000);
  zamanlayici.Interval = 30;
 zamanlayici.Enabled = True;
  anaForm.AddNewEvent(zamanlayici,tbeOnTimer,'sensorDurum');

  anaForm.Run;
}
```

11.3. Dokunma Sensörü

Gesture (hareket) olayı, bir bileşen (image veya form gibi) üzerindeki belirli bir hareket veya eylem tarafından tetiklenen olayı ifade etmektedir. Bu tür olaylar, kullanıcıların dokunmatik ekran kullanarak hareket ettirerek, yakınlaştırarak, döndürerek veya diğer dokunma hareketlerini gerçekleştirerek belirli bir bileşenle etkileşime girmesine olanak tanımaktadır.

Örneğin, bir mobil uygulamada, bir görüntüyü yakınlaştırmak için kullanıcı iki parmağını ekrana ayırarak bir "sıkıştırma" hareketi gerçekleştirebilir. Böyle bir hareket algılandığında ilgili bileşenin *tbeOnGesture* olayı tetiklenmektedir.

Bu tür etkinlikler dokunmatik ekranlı cihazlarda ve mobil uygulamalarda yaygın olarak kullanılmaktadır. Bu olayların işlenmesi, kullanılan programlama diline veya geliştirme platformuna bağlı olarak değişebilir.

Clomosy üzerinde birden fazla dokunma özellikleri kullanılabilir. Bu özellikleri uygulama üzerinden tanımlarken *clSetTouchIG* fonksiyonu kullanılmaktadır.

Özellik	**Kullanım Amacı**
igLongTap	Kullanıcının dokunmatik cihaza uzun süre basması ve basılı tutması durumunda tetiklenen bir olaydır. Bu olay, ekrana belirli bir süre boyunca dokunulduğunun algılanması için kullanılır. (264)
igZoom	Kullanıcıların resim gibi içeriği yakınlaştırmasına veya uzaklaştırmasına olanak tanıyan bir özelliği ifade eder.(259)
igRotate	Nesneyi döndürme işlemini ifade eder.(261)
igPan	Genellikle bir program veya platform içindeki içeriğin kaydırılması eylemi anlamına gelir. Bu eylem, içeriğin yukarı, aşağı, sola veya sağa gibi belirli bir yönde hareket etmesini sağlar.(260)

Örnek kullanım:

Form üzerinde kullanılacak dokunma hareketi birinci parametrede, hangi bileşen üzerinde dokunma hareketi yapılacağı ise ikinci parametre üzerinde gösterilmektedir.

```
MyForm.clSetTouchIG(igLongTap,GestureImg);
```

Bir form üzerinde dokunma hareketleri kullanıldığında hangi hareketin kullanıldığına dair geri dönüş *clFormGestureEvent_GestureID* ile alınır. Bu değer, yakınlaştırma, döndürme vb. gibi belirli bir hareketin tanımlayıcısını içerir. *Integer* bir değer döndürmektedir.

Örnek olarak bir görselde ekrana uzun süre dokunulduğu GestureID değerini bir TclLabel nesnesi üzerinde gösteren kod yazalım.

```
var
  anaForm:TclForm;
  dokunmaLbl:TclLabel;
  dokunmaImg:TClImage;

void dokunmaDurum;
{
  dokunmaLbl.Text = 'GestureID: ' + IntToStr(anaForm.clFormGestureEvent_GestureID);
}

{
  anaForm = TClForm.Create(Self);
  dokunmaLbl = anaForm.AddNewLabel(anaForm, 'dokunmaLbl','--');
  dokunmaLbl.Align = alMostTop;
  dokunmaLbl.Height = 20;

  dokunmaImg = anaForm.AddNewImage(anaForm, 'dokunmaImg');
  anaForm.setImage(dokunmaImg,'https://clomosy.com/demos/bg.png');
  dokunmaImg.Align = alClient;

  anaForm.clSetTouchIG(igLongTap,dokunmaImg);
  anaForm.AddNewEvent(dokunmaImg, tbeOnGesture, 'dokunmaDurum');

  anaForm.Run;
}
```

11.4. Mouse Hareket Sensörü

Mouse hareketleri, bilgisayar kullanıcılarının bir mouse aygıtı kullanarak ekranda yaptıkları çeşitli fiziksel hareketleri ifade eder. Mouse, bilgisayar ekranındaki imleci hareket ettirmek ve tıklama, sürükleme vb. eylemleri gerçekleştirmek için kullanılır. Mouse hareketleri, bilgisayar kullanıcılarına etkileşimli bir arayüz sağlar.

TRObject programlama dilinde mouse işaretçisini bir bileşenin üzerine getirdiğinizde tetiklenen olayı kullanmak için aşağıdaki hareketlere ve amaçlara bakabilirsiniz:

Özellik	Kullanım Amacı
tbeOnMouseDown	Bir bileşene (bir button veya image gibi) mouse ile tıklandığında tetiklenen bir olaydır. Bu olay mouse tuşuna basıldığında meydana gelir.
tbeOnMouseUp	Bir bileşenin mouse ile tıklandıktan sonra serbest bırakılmasıyla tetiklenen bir olaydır. Bu olay, mouse düğmesinin bırakıldığı anı temsil eder.
tbeOnMouseMove	Mouse işaretçisini bir bileşenin üzerine getirdiğinizde tetiklenen bir olaydır. Bu olay mouse işaretçisini hareket ettirdiğinizde meydana gelir.

Örnek kullanımı aşağıdaki gibidir. AddNewEvent fonksiyonu içerisinde ilk parametre üzerinde işlem yapılacak olan bileşeni içerir, ikinci parametre TclBasisEvent özelliği eklenir. Üçüncü parametre ise tetiklendiğinde yapılacak olan işlevin yazılmasıdır.

```
MyForm.AddNewEvent(GestureImg, tbeOnMouseUp, 'onFormMouseUp');
```

X konumuna erişebilmek için *clSenderMousePosX* ve Y konumuna ulaşabilmek için *clSenderMousePosY* özellikleri kullanılır. Bu özellikler mouse hareketini anlık olarak alır. Formun içindeki anlık fare konumunu güncellemek veya kullanıcıya anlık fare konumu hakkında bilgi sunmak için *MyForm.clFormMousePosX* ve *MyForm.clFormMousePosY* kullanılabilir.

Örnek üzerinde üç farklı olay (event) özelliği kullanılmıştır. Öncelikle ana kod kısmına bir TclLabel ve TclImage bileşeni eklensin ve olay (event) tetiklenmesi için fonksiyonlar tanımlansın.

```
var
  anaForm:TclForm;
  durumLbl:TclLabel;
  hareketImg:TClImage;
{
  anaForm = TClForm.Create(Self);

  durumLbl = anaForm.AddNewLabel(anaForm, 'durumLbl','--');
  durumLbl.Align = alTop;
  durumLbl.Height = 20;

  hareketImg = anaForm.AddNewImage(anaForm, 'hareketImg');
  anaForm.setImage(hareketImg,'https://clomosy.com/demos/bg.png');
  hareketImg.Align = alCenter;
  hareketImg.Height = 300;
  hareketImg.Width = 300;

  anaForm.AddNewEvent(hareketImg, tbeOnMouseDown,'onFormMouseDown');
```

```
  anaForm.AddNewEvent(hareketImg, tbeOnMouseUp, 'onFormMouseUp');
  anaForm.AddNewEvent(hareketImg, tbeOnMouseMove,'onFormMouseMove');

  anaForm.Run;
}
```

Ana kodlar yazıldıktan sonra tetiklenecek olan *onFormMouseDown*, *onFormMouseUp* ve *onFormMouseMove* prosedürleri tanımlanmalıdır.

```
void onFormMouseMove;
{
  durumLbl.Text = IntToStr(anaForm.clFormMousePosX) + ','+
IntToStr(anaForm.clFormMousePosY);
  durumLbl.Text = IntToStr(anaForm.clSenderMousePosX) +
','+IntToStr(anaForm.clSenderMousePosY);
}

void onFormMouseDown;
{
  durumLbl.Text = IntToStr(anaForm.clSenderMousePosX) + ','+
IntToStr(anaForm.clSenderMousePosY);
}

void onFormMouseUp;
{
  durumLbl.Text = IntToStr(anaForm.clSenderMousePosX) +
','+IntToStr(anaForm.clSenderMousePosY);
}
```

11.5. Tarayıcı Sensörü

Barkod okutularak tespit edilen seri numarasının elektronik ortama aktarılmasını sağlayan sensördür. Bu aşamadan sonra iletilen bilgiler (bilgisayar veya terminal üzerinde) analiz edilerek sonuç yansıtılır, böylece veri girişi çok daha hızlı ve doğru bir şekilde gerçekleştirilir.

Barkod okutma işleminde *CallBarcodeReader* fonksiyonu kullanılır. Bu fonksiyon, içerisine bir parametre alır. Bu parametre okutulmuş olan barkod bilgisinin bir değişkene veya nesneye aktarılmasına sağlar.

Örnek kullanımı:

```
MyForm.CallBarcodeReader(testLabel);
```

Bir örnek üzerinde deneyelim. Örnekte bir button ve label ekleyelim ve buttona tıklandığında kamera özelliği açılır ve barkod okutulur. Ardından okutulan barkod label içerisine yazılır.

```
Var
  anaForm:TclForm;
  okutBtn: TclButton;
  barkodBilgiLbl : TclLabel;

void barkodOkut;
{
  anaForm.CallBarcodeReader(barkodBilgiLbl);
}

{
  anaForm=TclForm.Create(self);

  okutBtn=anaForm.AddNewButton(anaForm,'okutBtn','Barkodu okutunuz');
  okutBtn.TextSettings.Font.Size=40;
  okutBtn.Align = alCenter;
  okutBtn.Height = 50;
  okutBtn.Width = 150;
  okutBtn.Margins.Bottom=200;
  barkodBilgiLbl=anaForm.AddNewLabel(anaForm,'barkodBilgiLbl','---');
  barkodBilgiLbl.Align = AlTop;
  barkodBilgiLbl.Width = 100;
  barkodBilgiLbl.Height =50;
  barkodBilgiLbl.Margins.Top=200;

  anaForm.AddNewEvent(okutBtn,tbeOnClick,'barkodOkut');

  anaForm.Run;
}
```

BÖLÜM 12. Animasyon ve Geçiş Efektleri

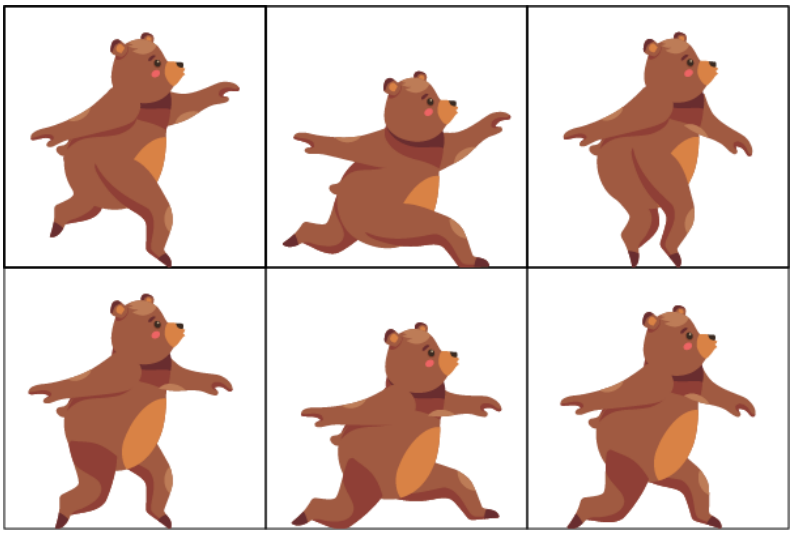
Kullanıcı ara yüzünde kullanılan nesne ve grafiklerin canlılığı ve esnekliği uygulamanın daha etkileşimli olmasını sağlamaktadır. Bu durum kullanıcıların dikkatini daha çok çekerek yapılan işe daha fazla odaklanmasını sağlamaktadır.

Konu kapsamında nesne veya herhangi bir bileşene nasıl hareket kazandırılacağı, bir efektin nasıl uygulanacağı ve bileşen üzerindeki bu etkilerin nasıl kaldırılacağı veya sonlandırılacağı anlatılacaktır.

Clomosy platformu üzerinde animasyon kullanımını gerçekleştirebilmek için *TclBitmapListAnimation* nesnesi kullanılmalıdır. Bu nesne ile animasyon görselinin dosya yolu verilir ve animasyon özellikleri eklenerek kullanılır. Bu nesnenin bir *TclRectangle* veya *TclCircle* bileşenlerinin tanımlanması ve içerisinde kullanılması gerekmektedir.

Animasyon özelliğini kullanabilmek için form olarak *TclGameForm* kullanılmaktadır. Diğer formlar için bu özellik kullanılamaz.

Bir örnek üzerinde animasyon nasıl yapılır bakalım.



Resim 66

Örnekte yukarıdaki görsel seçilmiştir. Öncelikle *TclBitmapListAnimation* oluşturulur. Bu nesne oluşturulurken *AddNewBitmapListAnimation* fonksiyonu kullanılır. Bu fonksiyon iki parametre alır. İlk parametre tanımlanacak olan nesnenin hangi bileşen içerisinde konumlandırılacağını belirler. İkinci parametre nesnenin ismini temsil eder.

```
var
  AnaForm:TclGameForm;
  animasyon1:TclBitmapListAnimation;
  circle1: TclCircle;

{
  AnaForm = TCLGameForm.Create(Self);
  AnaForm.AddGameAssetFromUrl('https://clomosy.com/demos/beardance_1.png');

  circle1 = AnaForm.AddNewCircle(AnaForm, 'circle1');
  circle1.Fill.Kind = fbkBitmap;
  circle1.Width = 200;
  circle1.Height = 200;
  circle1.Stroke.Thickness = 1;

  animasyon1 = AnaForm.AddNewBitmapListAnimation(circle1,'animasyon1');
  animasyon1.AnimationBitmap.LoadFromFile(clomosy.appfilespath + '/beardance_1.png');
  animasyon1.Enabled = True;

  AnaForm.Run;
}
```

Yukarıdaki kodda bir form oluşturulurdu. Form içerisine *TclCircle* nesnesi içerisine animasyon için *TclBitmapListAnimation* nesnesi oluşturulmuştur. Bu kod çalıştırıldığında ekranda sadece yuvarlak bir nesne içerisinde görselin ilk karesi yer alacaktır.

Animasyon kullanımı için adım adım neler yapılmalı bakalım.

Enabled

Animasyon aktifleştirilme işlemi gerçekleştirilir. Animasyon default (varsayılan) değeri *False* olarak tanımlanmıştır. Erişim sağlanması için *True* olarak atanmalıdır.

```
animasyon1.Enabled = True;
```

AnimationBitmap

Animasyon görselinin dosya yolu belirtilerek seçilir. Örnekte, *clomosy.Appfilespath* ile başlayan ve ardından '\beardance_1.png' ile tamamlanan bir dosya yolunu belirterek animasyonun yüklendiği anlamına gelir.

```
animasyon1.AnimationBitmap.LoadFromFile(clomosy.Appfilespath + '\beardance_1.png');
```

AnimationCount

Animasyonun içerisinde toplam kaç görsel bulunduğu belirtilmektedir. Örnekte, 6 görsel bulunduğu için değer olarak 6 ataması yapılmalıdır.

```
animasyon1.AnimationCount = 6;
```

AnimationRowCount

Animasyon görseli olarak belirlenen resimdeki satır sayısı tanımlanmaktadır. Örnekte, görsel, 3x2 matris şeklinde olduğundan dolayı satır sayısı olarak 2 belirtiliyor.

```
animasyon1.AnimationRowCount = 2;
```

Delay

Animasyon efekti başlamadan önce geçecek süre *Delay* fonksiyonu ile belirlenir.

```
animasyon1.Delay = 0;
```

Duration

Her animasyon görseli arasında geçen süreyi belirtmek için kullanılır. Süre azaldıkça animasyon hızı artar. Örnekte 6 görsel olduğu için 6 görsel arasında belirlenen süre gerçekleşir.

```
animasyon1.Duration = 0.75;
```

Stop
Çalışan animasyon durdurulur.

```
animasyon1.Stop;
```

Start
Animasyonun başlaması sağlanır.

```
animasyon1.Start;
```

AutoReverse
Animasyonun sonuna gelindiğinde sondan başa doğru ilerlemesini sağlamak için *AutoReverse* fonksiyonu **True** olarak atanmalıdır. Daha yumuşak bir animasyon efekti oluşturur fakat yürüme animasyonu gibi bir animasyonda karakterin bir ileri bir geri yürümesini sağlar.

```
animasyon1.AutoReverse = True;
```

Loop
Animasyonu sonsuz döngüye sokar. **False** durumda iken animasyon sadece 1 kere çalışır ve durur.

```
animasyon1.Loop = True;
```

Aşağıda örnekte Resim 66'da bulunan görselin animasyon ile kullanımı mevcuttur.

```
var
  AnaForm:TclGameForm;
  animasyon1:TclBitmapListAnimation;
  circle1: TclCircle;

{
  AnaForm = TCLGameForm.Create(Self);

  AnaForm.AddGameAssetFromUrl('https://clomosy.com/demos/beardance_1.png');

  circle1 = AnaForm.AddNewCircle(AnaForm, 'circle1');
  circle1.Fill.Kind = fbkBitmap;
  circle1.Fill.Bitmap.WrapMode = fbwmTileStretch;
  circle1.Width = 100;
  circle1.Height = 100;
  circle1.Stroke.Thickness = 1;

  animasyon1 = AnaForm.AddNewBitmapListAnimation(circle1,'animasyon1');
  animasyon1.AnimationBitmap.LoadFromFile(clomosy.appfilespath + '/beardance_1.png');
  animasyon1.Stop;
  animasyon1.AnimationCount = 6;
```

```
  animasyon1.AnimationRowCount = 2;
  animasyon1.Delay = 0;
  animasyon1.Duration = 0.75;
  animasyon1.AutoReverse = True;
  animasyon1.Enabled = True;
  animasyon1.Loop = True;
  animasyon1.Start;

  AnaForm.Run;
}
```

BÖLÜM 13. Sistem Fonksiyonları

Programlama dilleri, genellikle işletim sistemi ile etkileşimde bulunmak için özel sistem fonksiyonları veya kütüphaneleri içermektedir. TRObject dilinde yazılmış programların, özel sistem fonksiyonlarına erişimini sağlamak için kullanılan genel bir kavramdır. Bu fonksiyonlar, dilin standart kütüphanesinde veya platforma özgü kütüphanelerde bulunmaktadır.

Örneğin, Clomosy platformu üzerinde bir bileşenin özelliklerinde değişiklik yapılması gerektiğinde hazır fonksiyon olarak *ClRTSetProperty* fonksiyon kullanılmaktadır.

13.1. ClRTGetProperty

Uygulamanın çalışma zamanında, bir bileşenin belirli bir özelliğinin değerini döndürmektedir. Bu fonksiyon iki parametre almaktadır. İlk parametre kullanılacak bileşen adının, ikinci parametre seçilen bileşen özelliğinin yazılmasıdır.

```
clRTGetProperty(PropertyObject: TclComponent; PropName: String)
```

Örneğin bir formun içerisindeki bileşen adedini döndürmek için formun *ComponentCount* özelliğinin çağrılması gerekmektedir.

```
var
  anaForm : TclForm;
  adet : Integer;

{
  anaForm =TclForm.Create(self);
  adet = clRTGetProperty(anaForm, 'ComponentCount');

  ShowMessage('Form bileşen adedi: '+IntToStr(adet));

  anaForm.Run;
}
```

Bunun dışında button bileşeninin visible değeri, text özelliği gibi başka örneklerde verilebilir.

```
clRTGetProperty(Btn, 'Text')
clRTGetProperty(Btn, 'Visible')
```

13.2. ClRTSetProperty

Bir bileşenin özelliğini ayarlamak için kullanılmaktadır. Bu işlev, çalışma zamanında bir bileşenin belirli bir özelliğine bir değer atamanıza olanak tanımaktadır. Bu fonksiyon üç parametre almaktadır. İlk parametre kullanılacak bileşen adını, ikinci parametre seçilen bileşen özelliğini, üçüncü parametre ise özelliğin yeni değerinin yazılmasıdır.

```
clRTSetProperty(PropertyObject: TclComponent; PropName: String; PropValue: TclValue);
```

Örnek olarak bir butona tıklandığında butonun metninin *ClRTSetProperty* özelliği ile değiştirilmesini sağlayalım.

```
var
  anaForm : TclForm;
  bilgiBtn : TCLProButton;

void bilgiBtnGit;
{
  clRTSetProperty(bilgiBtn,'Text','Merhaba CLOMOSY!');
}

{
  anaForm = TclForm.Create(self);
  anaForm.SetFormBGImage('https://clomosy.com/demos/bg1.png');

  bilgiBtn = anaForm.AddNewProButton(anaForm,'bilgiBtn','Merhaba!');
  bilgiBtn.Align = AlTop;
  bilgiBtn.Height = 150;
  bilgiBtn.Margins.Top = 40;
  bilgiBtn.Margins.Left = 5;
  anaForm.AddNewEvent(bilgiBtn,tbeOnClick,'bilgiBtnGit');

  anaForm.Run;
}
```

13.3. ClGetStringReplace

Clomosy'de, bir metindeki karakteri veya alt dizeyi başka bir karakter veya alt dizeyle değiştirmek için *clGetStringReplace* işlevi kullanılmaktadır. *clGetStringReplace*, metinde arama ve istenilen değişiklikleri yapmak için kullanışlı bir işlevdir.

Bu fonksiyon üç parametre almaktadır. İlk parametre üzerinde işlem yapılacak olan metni, ikinci parametre değiştirilecek olan eski metni, üçüncü parametre ise eski metnin yerine gelecek olan yeni metnin veya karakterin yazılmasıdır.

clGetStringReplace(mainText,oldText,newText:String):String

Aşağıdaki örnekte bir metinde ad ve soyadı birleşik olarak alınıp arasında başka bir karakter olan metni, araya boşluk gelecek şekilde arama işlemi gerçekleştirilecektir.

```
var
  metinStr,eskiMtn,yeniMtn : String;
{
  metinStr = 'Ad%Soyad';
  eskiMtn = '%';
  yeniMtn = ' ';
  ShowMessage('Metin eski hali: '+metinStr);

  metinStr = clGetStringReplace(metinStr,eskiMtn,yeniMtn);

  ShowMessage('Metin yeni hali: '+metinStr);
}
```

13.4. ClGetStringTo

Metindeki bir karakter veya karakterlerden önce gelen metni almak için *clGetStringTo* işlevi kullanılmaktadır. Bu fonksiyon iki parametre almaktadır. Yani bir metin ve bir ayırıcı alır. İlk parametre üzerinde işlem yapılacak olan metni, ikinci parametre ise ayırıcı metnini almaktadır. Fonksiyon ayırıcıya kadar olan kısmı geri döndürmektedir. Eğer ayırıcı metinde bulunmazsa, tüm metni geri döndürür.

clGetStringTo(mainText, ConditionText'): String

Örnek üzerinde görelim.

```
var
  metinStr,ayiriciMtn : String;
{
  metinStr = 'Ad Soyad/TC';
  ayiriciMtn = '/';
  ShowMessage('Metin: '+metinStr);

  metinStr = clGetStringTo(metinStr,ayiriciMtn);

  ShowMessage('İşlevden sonrası: '+metinStr);
}
```

Bu örnekte basit bir mantık vardır. Girilen metnin sadece ad soyadı kısmı alınsın ve T.C. numarası alınmasın isteği gerçekleştirilmiştir.

13.5. ClGetStringAfter

Metindeki bir karakter veya karakterlerden sonra gelen metni almak için *ClGetStringAfter* işlevi kullanılmaktadır. Bu fonksiyon bir metin ve ayırıcı şeklinde iki parametre almaktadır. İlk parametre, üzerinde işlem yapılacak olan metni, ikinci parametre ise ayırıcı metni almaktadır. Fonksiyon ayırıcıdan sonraki kısmı geri döndürmektedir. Eğer ayırıcı metinde bulunmazsa, tüm metni geri döndürür.

```
clGetStringAfter(mainText, ConditionText'): String
```

Örnek olarak *clGetStringTo* işlevindeki örneğin aynısını yapalım. Bu sefer istenilen ayırıcı karakterden sonrasını ekrana getirecektir.

```
var
  metinStr,ayiriciMtn : String;
{
  metinStr = 'Ad Soyad/TC';
  ayiriciMtn = '/';
  ShowMessage('Metin: '+metinStr);

  metinStr = ClGetStringAfter(metinStr,ayiriciMtn);

  ShowMessage('İşlevden sonrası: '+metinStr);
}
```

13.6. ClStrToLan

clStrToLan fonksiyonu, uygulamada kullanılan cihazın dil ayarına göre çalışmaktadır. Fonksiyon string değer almaktadır. Ayırıcı olarak dikey uzun çizgi (|) kullanılarak işlem yapılmaktadır. Ayırıcının sol tarafındaki cümle İngilizce, sağ taraftaki ise Türkçe tanımlanmıştır.

Aşağıdaki örnekte bir butona tıklandığında butonun metnini cihazın diline göre değiştirmektedir.

```
var
  anaForm:TclForm;
  dilBtn : TclButton;

void cihazDilDestegi;
{
  dilBtn.Text = clStrToLan('Cihaz dili İngilizcedir.|Cihaz dili Türkçedir.');
}
{
```

```
  anaForm = TclForm.Create(Self);
  dilBtn= anaForm.AddNewButton(anaForm,'dilBtn','Cihaz dil desteğine göre mesaj
gönder...');
  dilBtn.TextSettings.Font.Size=20;
  dilBtn.Align = alCenter;
  dilBtn.Height = 50;
  dilBtn.Width = 300;

  anaForm.AddNewEvent(dilBtn,tbeOnClick,'cihazDilDestegi');

  anaForm.Run;
}
```

13.7. ClDoClick

Clomosy platformunda bir bileşen üzerinde programlı olarak tıklama işlemi gerçekleştirilmesini sağlamaktadır. Fonksiyon tek parametre almaktadır. Bu parametre bir bileşen alır.

Bu fonksiyon çağrıldığında belirtilen bileşen veya nesne üzerinde bir tıklama işlemi gerçekleştirilir. Bu, programın kullanıcı arayüzünü simüle etmek için veya otomasyon amacıyla kullanılabilir.

Örneğin, zamanlayıcı kullanılarak 5 saniye sonra görselin resmi değiştirilebilir. Bunun için görselin TclBasisEvent (*tbeOnClick*) tetiklemesi tanımlanmalıdır. Burada Img1Git prosedürü tetiklenmiştir. Bir de görsel üzerine tıklama işlemini iptal etmek için *HitTest* özelliği false olarak atanmaktadır.

```
var
  anaForm : TclForm;
  Img1 : TCLProImage;
  zamanlayici: TClTimer;

void Img1Git;
{
  Img1.clProSettings.PictureSource = 'https://clomosy.com/demos/ThumbsUp.png';
  Img1.SetclProSettings(Img1.clProSettings);
}

void zamanlayiciTetikle;
{
  clDoClick(Img1);
  zamanlayici.Enabled = False;
}
{
  anaForm=TclForm.Create(self);
  zamanlayici= anaForm.AddNewTimer(anaForm,'zamanlayici',5000);
```

```
  zamanlayici.Enabled = True;
  anaForm.AddNewEvent(zamanlayici,tbeOnTimer,'zamanlayiciTetikle');

  Img1 = anaForm.AddNewProImage(anaForm,'Img1');
  Img1.Align = AlTop;
  Img1.Height = 150;
  Img1.Margins.Top = 40;
  Img1.Margins.Left = 5;
  Img1.clProSettings.PictureSource = 'https://clomosy.com/demos/ThumbsDown.png';
  Img1.clProSettings.PictureAutoFit = True;
  Img1.SetclProSettings(Img1.clProSettings);
  anaForm.AddNewEvent(Img1,tbeOnClick,'Img1Git');
  Img1.HitTest = False;

  anaForm.Run;
}
```

13.8. GenerateRandom

Clomosy platformu üzerinde kullanılan özelleştirilmiş bir fonksiyondur. Bu fonksiyon iki sayı değeri arasında rastgele bir sayı üretmek için kullanılmaktadır. Rastgele üretilen sayıda fonksiyon içerisinde ilk parametre dahil, son parametre dahil değildir.

```
clMath.GenerateRandom(ilkDeger,sonDeger:Integer):Integer
```

Örnekte program çalıştırıldığı anda 10 ile 20 sayıları arasında rastgele bir sayı üretilecektir. Bu sayıyı bilmek için edit bileşenine bir sayı girilmelidir. Sayı doğru tahmin edilirse “tahmin doğru” mesajı verecek, yanlış tahmin edilirse “tahmin yanlış” diyerek doğrusunu gösterip işlem sona erecektir.

```
var
  anaForm:TclForm;
  tahminEdt : TclEdit;
  tahminBtn : TclButton;
  tahminSayisi : Integer;
void tahminEt;
{
  if(tahminEdt.Text == '')
  {
    ShowMessage('Sayı girin!')
  }else{
    if (StrToInt(tahminEdt.Text) == tahminSayisi){
      ShowMessage('Sayı doğru');
    }else{
      ShowMessage('Sayı yanlış. Doğru Sayı: '+IntToStr(tahminSayisi));
    }
```

```
  }
}
{
  anaForm = TclForm.Create(Self);
  tahminSayisi = clMath.GenerateRandom(10,20);

  tahminEdt= anaForm.AddNewEdit(anaForm,'tahminEdt','Tahmin ettiğin sayıyı gir...');
  tahminEdt.TextSettings.Font.Size=70;
  tahminEdt.Align = alTop;
  tahminEdt.Margins.Top= 10;
  tahminEdt.Height = 50;
  tahminEdt.Width = 150;

  tahminBtn= anaForm.AddNewButton(anaForm,'tahminBtn','Tahmin Et');
  tahminBtn.TextSettings.Font.Size=50;
  tahminBtn.Align = alCenter;
  tahminBtn.Height = 50;
  tahminBtn.Width = 100;
anaForm.AddNewEvent(tahminBtn,tbeOnClick,'tahminEt');

  anaForm.Run;
}
```

13.9. ClSender

Clomosy'de *ClSender* özelliği, bir programda tıklanan bileşene erişmeyi ve bu bileşene ait özellikler üzerinde değişiklik yapılmasını sağlamaktadır. Kısaca bir olay tetiklendiğinde, o olayla ilişkili nesnenin referansı *ClSender* özelliği aracılığıyla alınmaktadır.

Örneğin, birden fazla oluşturulan düğmenin OnClick olayında, tıklanan düğmeye erişmek için *ClSender* özelliği kullanılmaktadır. *ClSender* özelliği genellikle *TclObject* türündedir ve genel bir nesne başvurusunu temsil etmektedir. Ancak bu referansı belirli bir türe çevirerek o nesnenin özelliklerine ve yöntemlerine erişebilirsiniz.

```
Var
  anaForm:TclForm;
  anaPnl : TclProPanel;
  testBtn : TClProButton;
  i : Integer;
void BtnOnClick;
var
  clickedBtn:TClProButton;
{
  clickedBtn = TClProButton(anaForm.Clsender);
  clickedBtn.Caption = '+++'; //TClProButton(anaForm.Clsender).Caption
}
{
  anaForm=TclForm.Create(self);
```

```
  anaPnl=anaForm.AddNewProPanel(anaForm,'anaPnl');
  anaPnl.Align = AlCenter;
  anaPnl.Width = 200;
  anaPnl.Height = 300;
  anaPnl.clProSettings.IsRound = True;
  anaPnl.clProSettings.RoundHeight = 10;
  anaPnl.clProSettings.RoundWidth = 10;
  anaPnl.clProSettings.BorderColor = clAlphaColor.clHexToColor('#3a32a8');
  anaPnl.clProSettings.BorderWidth = 2;
  anaPnl.SetclProSettings(anaPnl.clProSettings);

  for (i = 0 to 4)
  {
    testBtn =
anaForm.AddNewProButton(anaPnl,'testBtn'+IntToStr(i+1),'testBtn'+IntToStr(i+1));
    testBtn.Align = AlTop;
    testBtn.Height = 50;
    testBtn.Margins.Top = 5;
    testBtn.Margins.Left = 5;
    testBtn.Margins.Right = 5;
    anaForm.SetImage(testBtn,'https://clomosy.com/demos/foodInformationBox.png');
    anaForm.AddNewEvent(testBtn,tbeOnClick,'BtnOnClick');
  }

  anaForm.Run;
}
```

13.10. ClFindComponent

Bir bileşen veya form altında başka bir nesne bulunması için *clFindComponent* kullanılmaktadır. Bu yöntem, belirli bir adlandırılmış alt bileşenin aranmasına olanak tanımaktadır. Bu, bulunan bileşenin özelliklerinde değişiklik yapılmasını sağlamaktadır.

```
MyForm.clFindComponent(xObjectName : String)
```

Bu özellik içerisinde bir string parametresi kabul edilir ve bu parametre değişkenin adıdır.

Örnekte *clSender* özelliği kullanılarak oluşturulan çoklu butonlardan hangisine tıklandığını belirleyebilir ve *clFindComponent* özelliği kullanılarak buton metni değiştirilmektedir.

```
Var
  anaForm:TclForm;
  anaPnl : TclProPanel;
  testBtn : TClProButton;
  i : Integer;
  nesneAdi:string;
void ismiGetir(gelenNesneAdi:String);
```

```
{
  TClProButton(anaForm.clFindComponent(gelenNesneAdi)).Caption = nesneAdi+'+';
}
void BtnOnClick;
{
  nesneAdi = TClProButton(anaForm.Clsender).Caption;
  ismiGetir(nesneAdi);
}
{
  anaForm=TclForm.Create(self);
  anaPnl=anaForm.AddNewProPanel(anaForm,'anaPnl');
  anaPnl.Align = AlCenter;
  anaPnl.Width = 200;
  anaPnl.Height = 300;
  anaPnl.clProSettings.IsRound = True;
  anaPnl.clProSettings.RoundHeight = 10;
  anaPnl.clProSettings.RoundWidth = 10;
  anaPnl.clProSettings.BorderColor = clAlphaColor.clHexToColor('#3a32a8');
  anaPnl.clProSettings.BorderWidth = 2;
  anaPnl.SetclProSettings(anaPnl.clProSettings);

  for (i = 0 to 4)
  {
    testBtn = anaForm.AddNewProButton(anaPnl,'testBtn'+IntToStr(i+1),
'testBtn'+IntToStr(i+1));
    testBtn.Align = AlTop;
    testBtn.Height = 50;
    testBtn.Margins.Top = 5;
    testBtn.Margins.Left = 5;
    testBtn.Margins.Right = 5;
anaForm.SetImage(testBtn,'https://clomosy.com/demos/foodInformationBox.png');

    anaForm.AddNewEvent(testBtn,tbeOnClick,'BtnOnClick');
  }

  anaForm.Run;
}
```

13.11. ClTagInt

ClTagInt özelliği herhangi bir nesnenin tamsayı değerini depolamak ve genellikle nesneye özgü ek bilgileri veya kimlik ayrıntılarını taşımak için kullanılmaktadır.

Aşağıda kullanımı mevcuttur:

```
textBtn.ClTagInt = 5;

deger = TclProButton(Myform.Clsender).ClTagInt; // deger : Integer
```

clSender bölümü üzerinde yapılan örneği biraz değiştirerek yazalım. Burada butonları oluştururken *clTagInt* özelliğine bir değer atanır.

```
for (i = 0 to 4)
  {
    testBtn = MyForm.AddNewProButton(testPanel,'testBtn'+IntToStr(i+1),'');
    testBtn.clTagInt = i+1;
    MyForm.AddNewEvent(testBtn,tbeOnClick,'BtnOnClick');
  }
```

Oluşturulan butona tıklandığında *clSender* özelliği kullanılarak *clTagInt* özelliğine erişim sağlanır ve hangi butona tıklandığı bilinmektedir.

```
void BtnOnClick;
{
  ShowMessage(TClProButton(Myform.Clsender).clTagInt);
}
```

Örnek kullanımı:

```
var
  anaForm : TclForm;
  panel1 : TclPanel;
  button1 : TclButton;
  i : Integer;

void BtnOnClick;
{
  ShowMessage(TclButton(anaForm.Clsender).clTagInt);
}

{
  anaForm = TclForm.Create(self);
  panel1 = anaForm.AddNewPanel(anaForm,'panel1');
  panel1.Align = ALLeft;
  panel1.Width = 200;

  for (i = 0 to 4)
  {
    button1 = anaForm.AddNewButton(panel1,'button'+IntToStr(i+1),'Button '+IntToStr(i+1));
    button1.Align = AlMostTop;
    button1.clTagInt = i+1;
    anaForm.AddNewEvent(button1,tbeOnClick,'BtnOnClick');
  }
  anaForm.Run;
}
```

13.12. CLRTMethod

Clomosy platformunda *CLRTMethod* fonksiyonu bir bileşenin birden fazla özelliklerine erişimi ve özelliklerin kullanılmasını sağlamaktadır. Bu fonksiyon iki parametre almaktadır. İlk parametre kullanılacak bileşen, ikinci parametreye ise erişilecek bileşenin özelliği yazılmaktadır.

clRTMethod(TclComponent, xMethod);

Bu fonksiyon üzerinde bir sürü özelliklere erişim sağlanabilir. Örnek olarak:

İki bileşen üst üste aynı konuma hizalandığında birinin önde görünmesi için *BringToFront* özelliği, arka kısımda kalması için *SendToBack* özelliği kullanılmaktadır.

Sadece Windows platformu üzerinde ve *clRTMethod* ile kullanılan bir diğer önemli işlev ise *ShowModal* metodudur. Bu metot *Run* görevini de yerine getirmektedir. Kullanıldığında *Run* metodu kullanılmamalıdır. Sonucunda hata alınır. *ShowModal* özelliği ile açılan form öncesindeki formlara erişim kapatılır ve gizlenir. Bu sayede Clomosy proje listesine tıklanmada engellenmiş olur.

Bunun dışında form nesnesinin çalıştırılması için Run, ekranda gösterilmesi için Show özellikleri de kullanılmaktadır.

```
clRTMethod(Btn1, 'BringTofront');
clRTMethod(Btn2, 'SendToBack');
clRTMethod(anaForm, 'Show');
clRTMethod(anaForm, 'Run');
clRTMethod(mainForm,'ShowModal'); //sadece Windows platformuna özel
```

13.13. ClFileExists

Clomosy platformunun bir parçasıdır ve belirtilen dosyanın varlığını kontrol eden bir işlevdir. Belirtilen dosya mevcutsa *True*, mevcut değilse *False* değerini döndürmektedir. Fonksiyon iki parametre almaktadır. İlk parametre, aranan dosya adının, ikinci parametre ise bu dosyanın yolunun string türünden yazılmasıdır.

clFileExists(FileNameString:WideString; CombinePathString:String);

Örnekte dosya yoluna ait konumda dosya bulunuyor ise dosya var değilse yok mesajı verilmektedir.

```
if (clFileExists('file.txt','D:\Clomosy'))
{
 clShowMessage('file exists');
}
Else
{
 clShowMessage('no file');
}
```

13.14. ClPathCombine

clPathCombine, Clomosy kütüphanesinin bir parçası olan bir fonksiyondur. Bu işlev, dosya yolunu ve dosya adını birleştirerek bir string veri döndürmektedir. İki parametre alan bu fonksiyonun ilk parametresi, dosya adını, ikinci parametre ise dosya yolunu almaktadır.

```
clPathCombine(FileNameString:WideString; CombinePathString:String);
```

Ardından bu fonksiyondan geri dönen değer, bir string veri türüne atanmaktadır.

Örneğin:

birlestirStr adında string bir veri oluşturulup bu veri içerisine *clPathCombine* fonksiyonundan gelen değer atanmaktadır. Ardından mesaj kutusunda birleştirilmiş hali gösterilmektedir. Eğer dosya uzantı içerisinde yok ise dosyayı oluşturur. Dosya var ise dosyaya yeni bir satır eklenir.

```
var
  strList:TclStringList;
  dosyaStr:String;
{
  strList = Clomosy.StringListNew;
  dosyaStr = clPathCombine('Dosya.Txt',Clomosy.AppFilesPath);
  ShowMessage('DOSYA YOLU:'+#13#10+dosyaStr);


  If clFileExists(dosyaStr,Clomosy.AppFilesPath)
  {
    strList.Add('Yeni satır 1');
    strList.SaveToFile(dosyaStr,0);
  } else
  {
    strList.SaveToFile(dosyaStr,0);
  }
}
```

13.15. clSaveToFile

Clomosy' de *clSaveToFile* prosedürü, bir bileşenin veya nesnenin verilerini bir dosyaya kaydetmek için kullanılan bir prosedürdür. Bu prosedür, verileri belirtilen dosya yoluna kaydederek, daha sonra bu dosyayı okumak veya başka bir amaç için kullanmak üzere verileri saklar. *clSaveToFile* prosedürü, iki parametre almaktadır. İlk parametre, dosya yolunu almaktadır. Bu genellikle bir dizin ve dosya adını içerir. Örneğin, 'C:\Users\Kullanıcı\Belgeler\cl.txt' gibi bir dosya yolu belirtilebilir. İkinci parametre, dosyaya kaydedilecek metin içeriğini oluşturur. Örneğin, 'Metin İçerik. Text değer' gibi bir metin içeriği belirtilebilir.

```
clSaveToFile(DosyaYolu,MetinIcerigi : String);
```

Aşağıdaki örnekte, bir memo bileşeni oluşturulmuştur. Bu nesne üzerine yazılan verilerin dosyaya kaydedilmesi için butona tıklanmalı ve kayıt işlemi yapılmalıdır.

```
var
  anaForm:TclForm;
  Memo1 : TclMemo;
  Layout1 : TclLayout;
  BtnEkle : TclProButton;
void BtnOnClick;
{
  ShowMessage(Clomosy.AppFilesPath);
  clSaveToFile(Clomosy.AppFilesPath+'cl.txt',Memo1.Text);
}

{
  anaForm = TclForm.Create(Self);

  Memo1 = anaForm.AddNewMemo(anaForm,'Memo1','');
  Memo1.Align = alMostTop;
  Memo1.Height = 200;
  Memo1.Width = 150;
  Memo1.Margins.Left= 10;
  Memo1.Margins.Right= 10;
  Memo1.Margins.Top= 10;
  Memo1.TextSettings.WordWrap = True;

  Layout1 = anaForm.AddNewLayout(anaForm,'Layout1');
  Layout1.Align=ALTop;
  Layout1.Height = 50;

  BtnEkle = anaForm.AddNewProButton(Layout1,'BtnEkle','');
  BtnEkle.Align = AlRight;
  BtnEkle.Width = 70;
  BtnEkle.Margins.Right = 10;
```

```
  anaForm.AddNewEvent(BtnEkle,tbeOnClick,'BtnOnClick');
  anaForm.SetImage(BtnEkle,'https://clomosy.com/demos/add-list.png');

  anaForm.Run;
}
```

13.16. clLoadFromFile

clLoadFromFile, TRObject programlama dilinde kullanılan bir fonksiyondur. Bu işlev, bir dosyadan metin veya veri okumak için kullanılır.

Bu işlev, okunan metni bir değişkene veya nesneye döndürmez. Bu nedenle, genellikle bir değişkene veya nesneye atama yapmak için kullanılmaz. Bunun yerine, doğrudan bir nesne özelliği veya metin içeren bir değişkene atanır.

Aşağıdaki örnekte, 'cl.txt' adlı dosyadan okunan metin, *Memo1* adlı bir *TclMemo* bileşeninin *Text* özelliğine atanır. Dosyanın tam yolu, *Clomosy.AppFilesPath* ile belirtilen bir dizine eklenir. Bu uygulamayı çalıştırmak için belirtilen dizine (*Clomosy.AppFilesPath*) 'cl.txt' adlı dosya oluşturulmalı, içerisine veriler eklenmelidir. Bu dizine istenilen dosya eklenmezse dosya bulunamadığı için *'Dosya belirtilen dizinde bulunamadı!'* mesajı alınacaktır.

```
var
  anaForm:TclForm;
  Memo1 : TclMemo;
  Layout1 : TclLayout;
 BtnEkle : TclProButton;

void BtnOnClick;
{
   ShowMessage(Clomosy.AppFilesPath);
   if (clFileExists('cl.txt',Clomosy.AppFilesPath))
   {
    Memo1.Text = clLoadFromFile(Clomosy.AppFilesPath+'cl.txt');
   }
   Else
   {
   clShowMessage('Dosya belirtilen dizinde bulunamadı!');
   }
}

{
  anaForm = TclForm.Create(Self);
  Memo1 = anaForm.AddNewMemo(anaForm,'Memo1','');
  Memo1.Align = alMostTop;
  Memo1.Height = 200;
  Memo1.Width = 150;
  Memo1.Margins.Left= 10;
  Memo1.Margins.Right= 10;
```

```
  Memo1.Margins.Top= 10;
  Memo1.TextSettings.WordWrap = True;

  Layout1 = anaForm.AddNewLayout(anaForm,'Layout1');
  Layout1.Align=ALTop;
  Layout1.Height = 50;

  BtnEkle = anaForm.AddNewProButton(Layout1,'BtnEkle','');
  BtnEkle.Align = AlRight;
  BtnEkle.Width = 70;
  BtnEkle.Margins.Right = 10;
  anaForm.AddNewEvent(BtnEkle,tbeOnClick,'BtnOnClick');
  anaForm.SetImage(BtnEkle,'https://clomosy.com/demos/add-list.png');

  anaForm.Run;
 }
```

13.17. clShowMessage

Clomosy yapısına ait mesaj kutusunu ifade etmektedir.

```
clShowMessage('Merhaba Dünya!');
```

13.18. Assigned

Assigned fonksiyonu, belirli bir değişkenin veya nesnenin bellekte bir değere sahip olup olmadığını kontrol etmek için kullanılmaktadır. Eğer belirtilen değişken veya nesne bir değere sahipse (bellekte bir nesne referansına sahipse), Assigned **True** değerini döndürür; değil ise **False** değerini döndürmektedir.

Aşağıdaki örnekte başlangıçta veri değişkenine değer atılmadan fonksiyona gönderilmektedir. Ardından veri atandıktan sonra tekrar fonksiyona gitmektedir. İlk mesaj ‘Değer boş!’, bir sonraki mesaj ise ‘Değer dolu!’ olarak gelecektir.

```
var
  veri : Integer
  mesaj: String;

function kontrol(gelenVeri:Integer) : String;
{
  if Assigned(gelenVeri)
    Result = 'Değer dolu!';
  else
    Result = 'Değer boş!';
}
```

```
{
  mesaj = kontrol(veri);
  ShowMessage(mesaj);
  veri = 20;
  mesaj = kontrol(veri);
  ShowMessage(mesaj);
}
```

13.19. HoldScreen

HoldScreen özelliği, ekran kilidinin açık veya kapalı olmasını belirler. Eğer *HoldScreen* özelliği *True* olarak ayarlanırsa, ekran kilidinin aktif olması engellenir ve kullanıcı form veya bileşen üzerindeki etkileşime devam edebilir. Eğer *False* olarak ayarlanırsa, ekran kilidi aktif hale gelir ve kullanıcı form veya bileşen üzerindeki etkileşimini durdurabilir.

Bu özellik genellikle bir uygulama, belirli bir işlem sırasında kullanıcının ekranın kilitlenmesini istemiyorsa veya kullanıcının bir süre boyunca etkileşimde bulunmaması gerekiyorsa kullanılır. Örneğin, bir oyun veya bir medya oynatıcı uygulaması, oyun içinde veya video izlerken ekran kilidini devre dışı bırakmak için *HoldScreen* özelliğini kullanabilir.

Varsayılan olarak, *HoldScreen* özelliği *False* olarak ayarlanır, bu da ekran kilidinin varsayılan olarak etkin olduğunu ve kullanıcının etkileşimi durdurduğunda ekranın kilitleyeceğini belirtir.

Kullanımı aşağıdaki gibidir.

```
Clomosy.HoldScreen = False;
```

BÖLÜM 14. Clomosy Kütüphanesi

Programlama dilleri, yazılım geliştirme süreçlerini kolaylaştırmak ve tekrar kullanılabilirlik sağlamak amacıyla kütüphane kavramını kullanmaktadır. Bir kütüphane, belirli bir amacı gerçekleştirmek üzere önceden yazılmış modüllerin veya fonksiyonların bir koleksiyonudur.

Kütüphaneler genellikle belirli görevleri yerine getirmek üzere tasarlanmış fonksiyonlar veya modüller içermektedir. Bu fonksiyonlar, geliştiricilere zaman kazandırarak, belirli görevleri baştan yazmak yerine hazır çözümleri kullanma imkânı sunmaktadır.

Kütüphaneler, yazılım geliştirme sürecinde tekrar kullanılabilmektedir. Bir kütüphanedeki fonksiyonlar veya modüller, farklı projelerde veya aynı projenin farklı bölümlerinde kolayca kullanılabilir.

Clomosy kütüphanesi, alt yapıda bulunan hazır sınıfların özelliklerini, veri ve davranışlarını bir araya toplayarak karmaşık projelerin daha anlaşılır şekilde tasarlanmasını sağlamaktadır.

14.1. Firma Verilerine Erişim

Geliştiricinin projelerini geliştirdiği firmanın GUID bilgisini string veri türünde döndürmektedir. Freemium hesapların firma GUID'si yoktur. Bu nedenle *CLOSTARTER* değerini döndürür.

Guid bilgileri benzersiz olduğundan ötürü bir başka geliştirici ile paylaşılmaması önerilir.

Kullanımı:

```
var
firma_GUID : String;
{
  firma_GUID = Clomosy.Firm_GUID;
  ShowMessage('Firma guid bilgisi: '+firma_GUID);
}
```

14.2. Proje Verilerine Erişim

Geliştirici ortamında oluşturulan projenin bazı bilgilerine erişim sağlanmaktadır.

Project_GUID

Kullanılan projenin GUID bilgisine erişim sağlanmaktadır. Projenin aktivasyon kodu geliştirici ortamında görülmektedir. Fakat GUID verisi gizli kalması için bu şekilde erişim sağlanır.

```
var
proje_GUID : String;
{
  proje_GUID = Clomosy.Project_GUID;
  ShowMessage('Proje guid bilgisi: '+proje_GUID);
}
```

AppProjectName

Kullanılan projenin adına erişim sağlanmaktadır.

```
var
proje_Adi : String;
{
  proje_Adi =  Clomosy.AppProjectName;
  ShowMessage('Proje adı: '+proje_Adi);
}
```

14.3. Platform Verilerine Erişim

Clomosy üzerinde geliştirilen projeler, Windows, iOS ve Android olmak üzere üç farklı platform üzerinde kullanılmaktadır. Platforma özgü özelliklere erişim sağlanabilmesi için kullanılan yöntemler bulunmaktadır. Hangi platformda çalıştırıldığı, cihaz dili gibi birden fazla farklı özellikler için kullanılan parametreler aşağıdaki gibidir.

PlatformIsMobile

Uygulamanın hangi platform üzerinde çalıştırıldığının bilgisi vermektedir. Uygulamanın açıldığı platform mobil ise *True*, değil ise *False* değerini döndürmektedir.

```
var
mobilMi : Boolean;
{
  mobilMi = Clomosy.PlatformIsMobile;
  if mobilMi
    ShowMessage('Uygulama mobil platformu üzerinde açıldı.');
  else
    ShowMessage('Uygulama windows platformu üzerinde açıldı.');
}
```

PlatformIsTurkish

Sistem diline erişim sağlamaktadır. Cihazın dili Türkçe ise *True* değerini döndürür; aksi takdirde *False* değerini döndürmektedir.

```
var
mobilMi : Boolean;
{
  mobilMi = Clomosy.PlatformIsTurkish;
  if mobilMi
    ShowMessage('Uygulama dili Türkçe"dir.');
  else
    ShowMessage('Uygulama dili İngilizce"dir.');
}
```

AppPlatform

Projenin açıldığı platformun geri dönüşü sağlanmaktadır. 0 değeri döndürülürse Windows, 1 ise MacOS, 2 ise IOS, 3 ise Android, 4 ise WinRT, 5 ise Linux ortamında açıldığını göstermektedir.

```
var
platformDurum : Integer;
{
  platformDurum = Clomosy.AppPlatform;
  case platformDurum of
  {
    0 : ShowMessage('Windows');
    1 : ShowMessage('MacOS');
    2 : ShowMessage('IOS');
    3 : ShowMessage('Android');
    4 : ShowMessage('WinRT');
    5 : ShowMessage('Linux');
  }
}
```

ClomosyID

Clomosy için 2 farklı mobil uygulaması bulunmaktadır. *Clomosy CRM* ve *Clomosy Learn* uygulamalarıdır. Hangi uygulamayı kullandığınızı görebilmek için bu özellik kullanılmaktadır. Dönen değerin 0 olması *Clomosy CRM* üzerinden projeye girildiği, 1 olması ise *Clomosy LEARN* üzerinden projeye girildiği anlamına gelmektedir.

```
var
uygulamaDurum : Integer;
{
  uygulamaDurum = Clomosy.ClomosyID;
  case uygulamaDurum of
  {
    0 : ShowMessage('Clomosy CRM');
    1 : ShowMessage('Clomosy Learn');
  }
}
```

CallUserProfile

Clomosy platformunda bir form oluşturulduğunda hazır bileşen yapıları da gelmektedir. Uygulamanın sol üst köşesinde geri butonu, sağ üst köşesinde menü butonu bulunmaktadır. Bu konu ‘Bölüm 9 Formlar’ kısmında anlatılmaktadır.

CallUserProfile özelliği ile geliştiricinin özel oluşturduğu bileşene profil sayfasının çağrılması sağlanmaktadır. Aradığınız yapıda profil sayfası açılıyor ve yan menüden geliyor.

```
var
anaForm:TclForm;
profilBtn : TclButton;

void goster{
  Clomosy.CallUserProfile(anaForm);
}
{
  anaForm = TclForm.Create(Self);
  profilBtn= anaForm.AddNewButton(anaForm,'profilBtn','Click');
  profilBtn.TextSettings.Font.Size=50;
  profilBtn.Align = alCenter;
  profilBtn.Height = 50;
  profilBtn.Width = 100;

  anaForm.AddNewEvent(profilBtn,tbeOnClick,'goster');

  anaForm.Run;
}
```

14.4. Kullanıcı Verilerine Erişim

Kullanıcı verilerine erişim, Clomosy platformu üzerinde platform, geliştirilen uygulamalar ve hizmet için temel bir gerekliliktir. Bu konuda sağlıklı ve güvenilir bir veri yönetimi hem kullanıcılara özelleştirilmiş deneyimler sunmak hem de geliştiricilere veri analizi yoluyla değerli bilgiler sağlamak açısından kritiktir.

Kullanıcı verilerine erişim, gizliliğin ve kişisel bilgilerin korunmasının öncelikli olduğu bir süreci içerir. Uygulamaların veya hizmetlerin kullanıcıya özelleştirilmiş içerikler, öneriler ve deneyimler sunmasını sağlar. Bu, kullanıcı memnuniyetini artırır.

Veri erişimi, kullanıcının onayı ve izni üzerine inşa edilir. Kullanıcılar, hangi bilgilerin paylaşılacağı konusunda kontrol sahibi olmalıdır. Hem mobil uygulamalarda hem de web tabanlı uygulamalarda geçerlidir. Bu, farklı platformlarda tutarlı bir kullanıcı deneyimi sağlamayı hedefler.

Clomosy platformu üzerinde geliştirici bir uygulama yazarken projesindeki kullanıcıların bilgilerine göre farklı arayüzleri tasarlayabilir ve sınırlama getirebilir. Bunun gibi farklı işlevleride gerçekleştirebilir.

AppUserGUID

Clomosy platformunda geliştirilen uygulamalar üzerinde, kullanıcılara göre farklı durumlar gerçekleştirilmektedir. Bunun için kullanıcının ID bilgisine ihtiyaç duyulmaktadır.

AppUserGUID fonksiyonu kullanılarak kullanıcının ID bilgisine erişim sağlanmaktadır. Fonksiyon string veri türünde geri dönüş yapmaktadır.

```
var
kullaniciID : String;

{
  kullaniciID = Clomosy.AppUserGUID;
  if kullaniciID == 'T2KJXXJR3'
    ShowMessage('Senin için yeni içerikler mevcuttur. İnceleyebilirsin.');
  else
    ShowMessage('Sizin için yeni bir içerik yoktur.');
}
```

AppUserDisplayName

Kullanıcının adı alınarak projelerde kullanıcıya özgü hitaplar ya da başka işlevler yapılabilir.

```
var
kullaniciAdi : String;

{
  kullaniciAdi = Clomosy.AppUserDisplayName;

  ShowMessage('Merhaba '+kullaniciAdi+'! Size nasıl yardımcı olabiliriz?');
}
```

AppUserProfile

Projeye kayıtlı olan üyeler arasında iki tip kullanıcı durumu mevcuttur. İki tip kullanıcıların biri yönetici yetkisi verilen kullanıcı, diğeri normal kullanıcı olarak geçmektedir. Bir kullanıcının nasıl yönetici yapılacağı Proje Üyeleri Bölümü üzerinde anlatılmaktadır.

Yönetici yetkisine sahip olan kullanıcılar, sistem üzerinde ayrıcalıklı yetkilere sahiptir. Normal kullanıcı ise yönetici ayrıcalıklarına sahip olmayan, sınırlı yetkilere sahip normal bir kullanıcıyı ifade etmektedir.

Uygulama üzerinde kullanıcının yönetici olup olmadığı *AppUserProfile* fonksiyonu ile öğrenilmektedir. Fonksiyon geri dönüş değeri integer veri türünde gelmektedir. Dönüş değeri 1 ise yönetici, 0 ise normal bir kullanıcıdır.

```
var
yoneticiBilgisi : Integer;
{
  yoneticiBilgisi = Clomosy.AppUserProfile;

  if (yoneticiBilgisi == 1)
    ShowMessage('Yönetici hesabına sahipsiniz.');
  else
    ShowMessage('Normal kullanıcı hesabına sahipsiniz.');
}
```

14.5. Unit Yönlendirme

RunUnit

Bir projede, proje dosyasına ilave olarak bir form dosyası ve bir kaynak kod dosyası oluşturulmaktadır. Bir birimi (unit) nasıl oluşturulacağı Unit Kullanımı bölümünden öğrenebilirsiniz. Bu dosyaya erişim sağlamak için birim (Unit) çağrılmaktadır. Bu işlem için *RunUnit* fonksiyonu kullanılmaktadır.

Aşağıdaki örnekte *uTest* adında oluşturulmuş birim (unit) çağrılmaktadır.

```
Clomosy.RunUnit('uTest');
```

14.6. Parametre Erişim

GetProjectUserDefParam

Projede global bir değişken oluşturulabilir ve bu projede istenilen bir unit üzerinde kullanılabilir. Ayrıntılı bilgi edinebilmek için Menu Araç Çubuğu başlığı altında anlatılmıştır. GetProjectUserDefParam fonksiyonunda bir parametre almaktadır. Bu parametre oluşturulan nesnenin adı yazılmalıdır. Alan adına erişim sağlamak için FieldByName diyerek istenilen alan adı yazılır ve bu şekilde kullanımı sağlanır.

Örnek tanımlaması aşağıdaki gibidir.

```
Clomosy.GetProjectUserDefParam('sinav_bilgileri').FieldByName('Value_Str').AsString;
```

14.7. Veri Tabanı Erişimi

Veri tabanları, büyük miktarda veriyi depolamak, yönetmek ve hızlı bir şekilde erişim sağlamak için kullanılan güçlü araçlardır. Veri tabanı erişimi, bu değerli veri havuzlarına güvenli ve etkili bir şekilde ulaşma yeteneğini ifade etmektedir. Veri tabanı erişimi, farklı işletim sistemleri ve platformlar arasında sorunsuz entegrasyonu desteklemektedir.

SQL Server, Lokal, Cloud ve JSON yapıları, farklı veri depolama ve yönetim yöntemlerini ifade eden terimlerdir. Bu terimler, farklı veri yönetim yaklaşımlarını ve depolama teknolojilerini temsil etmektedir. SQL Server, geleneksel bir ilişkisel veri tabanı sistemini temsil ederken, Lokal ve Cloud terimleri, veri tabanlarının konumlarına işaret etmektedir. JSON yapıları ise veri alışverişi ve depolama için kullanılan bir veri formatını ifade etmektedir. Clomosy üzerinde veri depolama işlemlerin yapılacağı farklı yapılar mevcuttur. Bunlara göz atalım.

14.7.1. SQL Server Bağlantıları

SQL Server, Microsoft tarafından geliştirilen bir ilişkisel veri tabanı yönetim sistemidir (RDBMS). SQL Server, veri tabanında veri depolamak, yönetmek, sorgulamak ve işlemek için

kullanılmaktadır. İlişkisel veri tabanı kavramına dayanır ve SQL (Structured Query Language) kullanılarak yönetilmektedir.

SQL bağlantısı sağlanmadan önce kullanılacak olan nesne tanımlanmalıdır. Bu nesnenin veri türü *TclSQLQuery* yapısında olmalıdır. Bu nesne tanımlaması yapıldıktan sonra Create işlemi yapılmalıdır. Ardından veri tabanı işlemleri gerçekleştirilir.

```
var
clDataQuery :TclSQLQuery;

{
  clDataQuery = TclSQLQuery.Create(Nil);
  …
}
```

DBSQLServerConnect

DBSQLServerConnect bir bağlantı nesnesini oluşturmak ve veri tabanına bağlanmak için kullanılan bir yöntemdir. Bu nesne, bağlantı bilgilerini içerir ve bu bilgilere dayanarak bir veri tabanı bağlantısı kurar. Yani, genellikle ilk başta kullanılır ve veri tabanına bağlanmak için gereken tüm bilgileri içermektedir.

```
Clomosy.DBSQLServerConnect(
'Veri tabanı Türü','Sunucu Adı','Kullanıcı Adı','Kullanıcı Şifresi','Veri tabanı Adı',port);
```

Bu fonksiyon altı parametre almaktadır. Bu parametrelerin amacı:

Veri tabanı Türü: Bağlanılmak istenen veri tabanının türünü belirtir. Bu durumda "SQL Server" olduğu yazılmalıdır.

Sunucu Adı: SQL Server veri tabanına bağlanılacak olan sunucunun adı veya IP adresi girilmelidir.

Kullanıcı Adı: Veri tabanına bağlanmak için kullanılacak kullanıcı adı girilmelidir.

Kullanıcı Şifresi: Kullanıcı adı ile eşleşen şifre girilmelidir.

Veri Tabanı Adı: Bağlanılacak olan veri tabanı adı girilir.

Port: SQL Server'ın kullanılan bağlantı noktası. Genellikle varsayılan SQL Server bağlantı noktası 1433'tür.

Aşağıda bir örnek bulunmaktadır. Veri tabanı gizliliğini sağlamak için rastgele veriler yazılacaktır.

```
void SetupSqlConnection;
{
  Clomosy.DBSQLServerConnect('SQL
Server','CloData','denemeKullanici','123456','Urunler',1433);
}

{
  SetupSqlConnection;
}
```

DBSQLServerConnection

DBSQLServerConnection fonksiyonu, bir bağlantı nesnesinin var olduğu durumlarda kullanılır. Yani, önceden oluşturulmuş bir bağlantı nesnesine (örneğin, *DBSQLServerConnect* tarafından oluşturulan nesneye) referans almak için kullanılır. Bu nesne, zaten kurulmuş ve yapılandırılmış bir bağlantıyı temsil eder.

```
var
clDataQuery :TclSQLQuery;

{
  clDataQuery = TclSQLQuery.Create(Nil);

  try
    clDataQuery.Connection = Clomosy.DBSQLServerConnection;
    ...
  finally
    ...
  }
}
```

DBSQLServerQueryWith

DBSQLServerQueryWith bir işlevdir ve global bağlantıyı (*DBSQLServerConnect*) kullanarak global sorguyu (query) gerçekleştirir. Bu işlev, sorguyu açar ve sonucunu döndürür.

```
Function DBSQLServerQueryWith(SQLStr:String):TclSqlQuery;
```

Fonksiyon bir parametre almaktadır. String veri türünde SQL sorgusu yazılmalıdır.

Kodun genel amacı, belirtilen tablodan belirli bir sütunu seçip, bu sütunu farklı bir adla kullanarak bir sorgu sonucu elde etmektir.

Aşağıdaki örnekte, bir SQL Server veri tabanına bağlanmayı, bir sorgu çalıştırmayı ve sorgu sonuçları üzerinde döngü ile işlem yapmayı göstermektedir.

VeriSorgusu adlı bir değişken kullanılarak sorgu sonucu alınır. Bu *VeriSorgusu* değişkeni, sorgu sonucundan dönen veriyi tutmak veya işlemek için kullanılmaktadır. Bu tür bir yapı, veri tabanı ile etkileşimde bulunmak ve uygulama içinde bu veriyi kullanmak amacıyla sıkça kullanılır.

```
var
VeriSorgusu : TClSQLQuery;

{
  Clomosy.DBSQLServerConnect('SQL Server', 'sunucu_adi', 'kullanici_adi',
'kullanici_sifre','veritabani_adi', 1433);

  try
    VeriSorgusu = Clomosy.DBSQLServerQueryWith('SELECT * FROM UrunBilgisi');
    VeriSorgusu.Open;
    while (not VeriSorgusu.Eof)
    {
      ShowMessage(VeriSorgusu.FieldByName('urun_adi').AsString);
      VeriSorgusu.Next;
    }
  finally
    VeriSorgusu.Free;
  }
}
```

DBSQLServerQuery

Veri tabanı işlemlerini gerçekleştirmek üzere kullanılan bir sorgu nesnesini ifade eder. Bu sorgu Clomosy SQL yapısının kullanılacağı bir diğer sorgu yapısıdır.

DBSQLServerQuery özelliği, global TclSqlQuery nesnesini temsil eder. *DBSQLServerConnect* kullanıldığında, bu nesneyi oluşturur ve bağlantısını kurar.

```
Clomosy.DBSQLServerQuery.Sql.Text = 'SELECT * FROM UrunBilgisi WHERE
urun_fiyati=101';
```

Bir örnek üzerinde öğrenilen bilgileri kullanalım.

Örnekte, bir SQL Server veri tabanına bağlanmayı, bir sorgu çalıştırmayı ve sorgu sonuçlarına göre veri tabanında işlemler yapmayı göstermektedir. Eğer sorgu sonucunda en az bir kayıt bulunmuşsa (RecordCount değeri 0'dan büyükse), bu kayıt üzerinde düzenleme yapmak için *Clomosy.DBSQLServerQuery.Edit* metodu çağrılır.

Eğer sorgu sonucunda hiç kayıt bulunmazsa, yeni bir kayıt eklemek için *Clomosy.DBSQLServerQuery.Insert* metodu çağrılır. Ardından, yeni kayıtın alanları belirlenir (id, isletim_sistemi, cihaz_tipi), ve *Clomosy.DBSQLServerQuery.Post* ile bu yeni kayıt veri tabanına eklenir.

Daha sonra, her iki durumda da *Clomosy.DBSQLServerQuery.FieldByName* ile belirli alanlara değerler atanır ('model' ve 'urun_fiyati'). Ardından, Clomosy.DBSQLServerQuery.Post ile yapılan değişiklikler veri tabanına kaydedilir.

except bloğu ile hata kontrolü yapılır: Eğer bir hata oluşursa (*try* bloğundaki işlemlerde bir hata meydana gelirse), *except* bloğu çalışır ve hatanın detayları *ShowMessage* ile ekrana yazdırılır.

```
{
  Clomosy.DBSQLServerConnect('SQL Server', 'sunucu_adi', 'kullanici_adi',
'kullanici_sifre','veritabani_adi', 1433);
  try
Clomosy.DBSQLServerQuery.Sql.Text = 'SELECT * FROM UrunBilgisi WHERE
urun_fiyati=103';
    Clomosy.DBSQLServerQuery.Open;

    If (Clomosy.DBSQLServerQuery.RecordCount > 0) //not
Clomosy.DBSQLServerQuery.Found
    {
      Clomosy.DBSQLServerQuery.Edit;

    } else
    {
      Clomosy.DBSQLServerQuery.Insert;
      Clomosy.DBSQLServerQuery.FieldByName('id').AsString = '13';
      Clomosy.DBSQLServerQuery.FieldByName('isletim_sistemi').AsString = 'MacOS';
      Clomosy.DBSQLServerQuery.FieldByName('cihaz_tipi').AsString = 'ipad';
      //Clomosy.DBSQLServerQuery.Sql.Text = 'INSERT INTO UrunBilgisi
(id,isletim_sistemi,cihaz_tipi) VALUES (12,'+QuotedStr('MacOS')+','+QuotedStr('ipad')+')';
      //Clomosy.DBSQLServerQuery.ExecSQL;
    }
      Clomosy.DBSQLServerQuery.FieldByName('model').AsString = 'Alpha';
      Clomosy.DBSQLServerQuery.FieldByName('urun_fiyati').AsInteger = 104;
      Clomosy.DBSQLServerQuery.Post;

  except
    ShowMessage('Exception Class: '+LastExceptionClassName+' Exception Message:
'+LastExceptionMessage);
  }
}
```

DBXCopyRecord

DBXCopyRecord fonksiyonu, iki dataset nesnesi arasında bir kaydı kopyalamak için kullanılır. Bu fonksiyon, iki parametre almaktadır. İlk parametre kaynak sorgu yapısını döndürür, ikinci parametre ise hedef sorgudur. Hedef sorguya, kaynakdan alınan sorgu kopyalanmaktadır.

```
procedure DBXCopyRecord(SourceQ, DestQ: TclDataSet);
```

Örnekte olduğu gibi, *kaynakSorgu* adlı bir *TClSqlQuery* nesnesi ile bir SQL sorgusu çalıştırılır ve bir kayıt alınır. Daha sonra *hedefSorgu* adlı başka bir *TClSqlQuery* nesnesine bir başka SQL sorgusu ile ulaşılır.

DBXCopyRecord fonksiyonu, bu iki sorgu nesnesi arasındaki kaydı kopyalar. Bu işlem genellikle bir kaydın düzenlenmesi (Edit), kopyalanması (Clomosy.DBXCopyRecord), ardından kopyalanan kaydın Post edilmesi işlemlerini içerir. Yani, *kaynakSorgu* nesnesinden alınan kayıt, *hedefSorgu* nesnesine kopyalanır ve ardından Post edilerek kayıt değişiklikleri onaylanır.

i	Unutmayın! Kopyalama işlemi yapılacak durumlarda tabloların alanlarının birbirine eşit ve alan adlarının aynı olması gerekmektedir.

```
var
  kaynakSorgu, hedefSorgu: TClSqlQuery;
{
  kaynakSorgu = TClSqlQuery.Create(nil);
  hedefSorgu = TClSqlQuery.Create(nil);
  Clomosy.DBSQLServerConnect('SQL Server', 'sunucu_adi', 'kullanici_adi',
'kullanici_sifre','veritabani_adi', 1433);

  try
    kaynakSorgu.Connection = Clomosy.DBSQLServerConnection;
    hedefSorgu.Connection = Clomosy.DBSQLServerConnection;
    kaynakSorgu.SQL.Text = 'SELECT id,ad as urun_adi, fiyat as urun_fiyati FROM
KargoUrunBilgileri WHERE fiyat = 125';
    kaynakSorgu.Open;
    hedefSorgu.SQL.Text = 'SELECT * FROM UrunBilgileri WHERE 1=0';
    hedefSorgu.Open;
    hedefSorgu.Edit;

    Clomosy.DBXCopyRecord(kaynakSorgu, hedefSorgu);

    hedefSorgu.Post;
  finally

    kaynakSorgu.Close;
    hedefSorgu.Close;
    kaynakSorgu.Free;
    hedefSorgu.Free;

  }
}
```

14.7.2. Local (Yerel) Veri Tabanı Bağlantıları

Local veri tabanları, genellikle kullanıldığı cihaz veya uygulama özelinde bulunan, genellikle tek bir bilgisayar veya cihazda çalışan veri tabanlarıdır. Bu veri tabanları, genellikle kullanıcının bilgisayarında veya bir uygulama içinde depolanır ve yönetilir.

DBSQLiteConnect

Clomosy, bir SQLite veri tabanı bağlantısı oluşturmak için *DBSQLiteConnect* fonksiyonunu kullanır. Bu fonksiyon, iki string parametresi alır: birincisi veri tabanı dosyasının yolu, ikincisi ise veri tabanının şifresidir. Aşağıdaki örnek, bu fonksiyonun kullanımını göstermektedir:

```
Clomosy.DBSQLiteConnect('dosyaYolu', 'sifre');
```

Örnek kullanımı:

Aşağıdaki örnekte, uygulama Windows'ta açıldığında, C klasörü içerisinde önceden oluşturulmuş ‘sqlite_db’ klasörüne ‘yeni.db3’ adında yeni bir veri tabanı oluşturulmaktadır. Mobil cihazlarda uygulama açıldığında ise, uygulamanın dosya yoluna ‘yeni.db3’ adında bir veri tabanı oluşturulur.

```
var
  Sifre,DB  : String;

{
  Sifre = '';
  if (Clomosy.PlatformIsMobile)
  {
   DB = Clomosy.AppFilesPath + 'yeni.db3';
  }else
  {
   DB = 'C:\sqlite_db\'yeni.db3';
  }
  Clomosy.DBSQLiteConnect(DB, Sifre);
  ShowMessage('Veri tabanı başarılı bir şekilde oluştu.');
}
```

DBSQLiteQueryWith

DBSQLiteQueryWith fonksiyonu, Clomosy platformunda SQLite veri tabanı sorgularını oluşturmak ve çalıştırmak için kullanılır. Bu fonksiyon, bir SQL sorgusuyla ilişkilendirilmiş yeni bir *TClSqlQuery* nesnesi oluşturur ve bu nesneyi geri döndürür. Oluşturulan *TClSqlQuery* nesnesi, veri tabanı sorgularını çalıştırmak ve sonuçları almak için kullanılabilir.

Bu fonksiyonun temel amacı, SQLite veri tabanı işlemleri için bir sorgu nesnesi oluşturmak ve bu nesneyi kullanarak sorguları gerçekleştirmektir. Örneğin, bir SELECT sorgusu yapmak ve sonuçları almak için kullanılabilir.

Bu fonksiyon bir parametre almaktadır. Bu parametre string veri türünde olup SQL sorgusunu içermektedir.

Örnek Kullanımı:

Aşağıdaki örnekte uygulamaya geçmeden önce ‘okul.db’ adında bir veri tabanı oluşturuldu. Bu veri tabanı içerisinde ‘ogrenci’ adında bir tablo oluşturuldu. Uygulama, ‘ogrenci’ tablosunda veri bulunup bulunmadığını kontrol eder. Eğer veri varsa, bir döngü yardımıyla ‘ad’ alanındaki veriler döndürülür.

```
var
  Sifre,DB  : String;
  Qry : TClSQLiteQuery;
{
  Sifre = '';

  DB = 'C:\sqlite_db\okul.db';

  Clomosy.DBSQLiteConnect(DB, Sifre);

   try
    Qry = Clomosy.DBSQLiteQueryWith('SELECT * FROM ogrenci');
    Qry.OpenOrExecute;
    while (not Qry.Eof)
    {
      ShowMessage(Qry.FieldByName('ad').AsString);
      Qry.Next;
    }
  except
   ShowMessage('Exception Class: '+LastExceptionClassName+' Exception Message:
'+LastExceptionMessage);

  }
}
```

DBSQLiteQuery

DBSQLiteQuery ifadesi, Clomosy platformunda SQLite veri tabanı işlemleri için SQL sorgusunu belirlemek amacıyla kullanılır. Bu ifade, *TClSqlQuery* nesnesinin SQL sorgusunu belirlemek için kullanılan bir özelliktir.

Örnek kullanımı:

Bu örnekte, Clomosy platformu üzerinde SQLite veri tabanı kullanarak basit bir sorgu gerçekleştiriliyor. Sorguda bir veri var ise sadece ilk kaydın soyadı alan bilgisi ekrana gösterilir.

```
var
  Sifre,DB  : String;
  Qry : TClSQLiteQuery;
{
  Sifre = '';

  DB = 'C:\sqlite_db\okul.db';
  Clomosy.DBSQLiteConnect(DB, Sifre);

   try
    Clomosy.DBSQLiteQuery.Sql.Text = 'SELECT * FROM ogrenci';
    Clomosy.DBSQLiteQuery.OpenOrExecute;
    If (Clomosy.DBSQLiteQuery.Found)
       ShowMessage(Clomosy.DBSQLiteQuery.FieldByName('soyad').AsString);

  except
   ShowMessage('Exception Class: '+LastExceptionClassName+' Exception Message:
'+LastExceptionMessage);

  }
}
```

14.7.3. Cloud (Bulut) Veri Tabanı Bağlantıları

Cloud veri tabanları, bulut bilişim hizmetleri aracılığıyla sağlanan ve yönetilen veri tabanlarıdır. Bu tür veri tabanları, çeşitli bulut hizmet sağlayıcıları tarafından sunulur ve genellikle ölçeklenebilir, yedeklenebilir ve erişimi kolaydır. Kullanıcılar, bu veri tabanlarına internet üzerinden erişebilirler.

Clomosy üzerinde Premium hesaba sahip kullanıcılara atanmış hazır tablolar bulunmaktadır. Bu tablolara Şablonlar (Templates) denilmektedir.

Platformun birçok sunmuş olduğu tablolar bulunmaktadır. Bu tabloların kendilerine ait alanları mevcuttur. Items, Products, Types, Employess, Customers,Tasks,Groups, SurExams ve Thread tabloları bulunmaktadır.

i	Clomosy hazır tablolar ve tablo alanlarına dair ayrıntılı bilgiler *Clomosy Şablon (Template) Yapısı* konu başlığı altında anlatılmıştır. Yapılan uygulamaya özgü istenilen tablo kullanılabilir.

DBCloudSQLSelectWith

DBCloudSQLSelectWith işlevi bir SQL sorgusunu yürütmek için kullanılır. Bu sorgunun *TCLJSONQuery* sınıfına ait sonuç kümesini temsil eden bir nesne oluşturur. Bu nesne sorgu sonuçlarına erişmek ve bunları değiştirmek için kullanılır. Bu fonksiyon bir parametre almaktadır ve string veri türünde olmalıdır.

```
Clomosy.DBCloudSQLSelectWith('SELECT * FROM tblProjectProducts')
```

Örnek kullanımı:

```
var
  productSorgu:TCLJSONQuery;
  qryStr:String;
{
  qryStr='SELECT * FROM tblProjectProducts';
  productSorgu = Clomosy.DBCloudSQLSelectWith(qryStr);
  with productSorgu do
  {
    First;
    if (Found)
    {
      ShowMessage(productSorgu.getJSONString);
      while (not EOF){
        ShowMessage(FieldByName('Product_Name').AsString);
        Next;
      }
    }
  }
}
```

Yukarıdaki örnekte *Products* tablosundaki veriler alınmakta ve içerisinde veri varsa *getJSONString* ile bütün veriler mesaj kutusunda gösterilir. Ardından döngü ile verilerin *Product_Name* alanı gösterilmektedir.

DBCloudQueryWith

DBCloudQueryWith yöntemi, belirli parametreler ile bir bulut veri tabanı sorgusu oluşturur. Bu yöntem, bulut veri tabanı ile etkileşimde bulunmak için kullanılan sorgu ile çalıştırılan bir yöntemdir.

Fonksiyon üç parametre almaktadır. İlk parametrede çağrılacak olan tablo adı yazılır, ikinci parametrede sorguya string veri türünde filtreleme işlemi gerçekleştirilir, son parametre de ise sorgu şartı verilir.

```
Clomosy.DBCloudQueryWith(ftMembers,'','1=1 ORDER BY NEWID()');
```

Örnek Kullanımı:

```
var
uyeSorgusu:TCLJSONQuery;
{

  try
```

```
    uyeSorgusu = Clomosy.DBCloudQueryWith(ftMembers,'','1=1 ORDER BY NEWID()');
//her seferinde karışık member listesi al
    uyeSorgusu.Filtered = False;
    uyeSorgusu.Filter = 'Member_GUID <> '+QuotedStr(Clomosy.AppUserGUID);
    uyeSorgusu.Filtered = True;

    if (uyeSorgusu.Found)
    {
      ShowMessage(uyeSorgusu.GetJSONString);
    while (not uyeSorgusu.EOF)
    {
      ShowMessage(uyeSorgusu.FieldByName('Member_Name').AsString);
      uyeSorgusu.Next;
    }
    }
  finally
    uyeSorgusu.Free;
  }
}
```

Örnekte *uyeSorgusu* adında *TclJSONQuery* nesnesi üzerine sorgu ile üyeler listesi çağrılmıştır. Sorguda şart olarak her seferinde rastgele değerlerin getirilebilmesi için ORDER BY NEWID() kullanılmıştır.

Üyeler listesi getirilirken uygulamayı kullanan kullanıcının dahil edilmemesi için filtreleme işlemi yapılmıştır. Sorgu üzerinde filtreleme işlemi için öncelikle filtereme işleminin aktifleştirilmesi sağlanmıştır ardından filtre ifadesi;

'Member_GUID < > '+QuotedStr(Clomosy.AppUserGUID)

Olarak ayarlanmıştır. Sorgu işleminden sonra eğer listede veri var ise *getJSONString* ile bütün veriler mesaj kutusunda gösterilir. Ardından döngü ile verilerin *Member_Name* alanı gösterilmektedir.

DBCloudPostJSON

DBCloudPostJSON fonksiyonu, Clomosy platformunda bulunan bir bulut veri tabanına, tabloda yapılan değişiklikleri güncellemek veya yeni kayıtlar eklemek için kullanılır.

```
Function DBCloudPostJSON(CloudDataSource:TFormTemplate; DataJSON:String):Boolean;
```

Aşağıdaki örnekte *ftMembers* tablosundaki uygulamaya giren kullanıcının id ve ad bilgileri alınarak ftThreads tablosuna yeni kayıt atılmaktadır.

```
var
uyeSorgusu:TCLJSONQuery;
uyeGUID,uyeAd :String;
{
 try
   uyeSorgusu =  Clomosy.DBCloudQueryWith(ftMembers,','PMember_GUID = '+
QuotedStr(Clomosy.AppUserGUID));
   if (uyeSorgusu.Found)
   {
     ShowMessage(uyeSorgusu.GetJSONString);
     uyeGUID = uyeSorgusu.FieldByName('PMember_GUID').AsString;
     uyeAd = uyeSorgusu.FieldByName('PMember_Name').AsString;
Clomosy.DBCloudPostJSON(ftThreads,'[{"Thread_Value_Text":"'+uyeGUID+'","Thread_
Value_Str":"'+uyeAd+'"}]');
   }
 finally
   uyeSorgusu.Free;
 }
}
```

DBCloudQuery

DBCloudQuery, Clomosy platformunda global bir sorgu tanımlama ve diğer birimlerde çağırma işlemlerini gerçekleştirebilen bir fonksiyondur. Bu fonksiyon, veri tabanındaki bilgilere erişim sağlamak için kullanılır ve projenin farklı birimlerinde kullanılmak üzere genel bir sorgu oluşturulmasına olanak tanır. Bu sayede, veri tabanı işlemleri daha modüler hale getirilebilir.

Bu fonksiyon, projenin farklı birimlerinde ihtiyaç duyulan veri tabanı sorgularını merkezi bir konumda tanımlamayı ve bu sorguları diğer birimlerden çağırmayı sağlar. Bu sayede, veri tabanı işlemleri daha düzenli ve yönetilebilir bir şekilde gerçekleştirilebilir.

Bir sorgu ataması yapabilmek için kullanılan örnek yapı:

```
Clomosy.DBCloudQuery =
Clomosy.DBCloudQueryWith(ftGroups,','ISNULL(Rec_Deleted,0)=0');
```

Oluşturulan sorgu aşağıdaki gibi kullanılır:

```
var
  globalSorgu:TCLJSONQuery;
{
  globalSorgu = Clomosy.DBCloudQuery;
}
```

14.7.4. Json Yapısı

JSON (JavaScript Object Notation), veri alışverişi için kullanılan bir hafif veri formatıdır. JSON, insanlar tarafından okunabilir ve yazılabilir olması yanında, programlar arası veri alışverişinde kullanılabilecek bir formata sahiptir. JSON yapıları, anahtar-değer çiftleri ve diziler içeren bir veri formatını ifade eder.

Json veri seti kullanımı için *TCLJSONQuery* sınıfına ait bir değişken oluşturulmalıdır.

```
var
  jsonVerisi:TCLJSONQuery;
```

Ardından değişkeni Create işlemi ile oluşturulabilir.

```
jsonVerisi = TCLJSONQuery.Create(nil);
```

ClDataSetFromJSON

ClDataSetFromJSON fonksiyonu, bir JSON dizisini alır ve bu diziyi bir dataset nesnesine dönüştürür. Bu, JSON formatındaki verileri Clomosy uygulamalarındaki dataset türünden veri setlerine dönüştürmek için kullanılır.

Bu fonksiyon bir parametre almaktadır. Bu parametre string veri türünde olmalıdır ve json formatına uygun olmalıdır.

JSON formatındaki veriler, genellikle web servislerinden veya dış kaynaklardan alınan verileri temsil eder. *ClDataSetFromJSON* fonksiyonu, bu verileri Clomosy'nin veri seti yapılarına uygun bir formata çevirir, böylece bu veriler kolayca kullanılır.

Örnek bir json dizisi kullanımı:

```
Clomosy.ClDataSetFromJSON('[{"ad":"İsmail","yas":27},{"ad":"Tuğba","yas":25}]');
```

DBDatasetGetAsJSON

Bu *DBDatasetGetAsJSON* fonksiyonu, bir *dataset* türündeki veri setini JSON formatına dönüştürerek elde edilen JSON dizesini döndürür. Bu tür bir fonksiyon genellikle veri setinde bulunan verileri JSON formatında bir servise göndermek, bir dosyaya yazmak veya başka bir uygulamaya iletmek gibi senaryolarda kullanılır. Bu fonksiyonun geri dönüş değeri string'dir.

```
Function DBDatasetGetAsJSON(SourceQ: TclDataSet):String;
```

Örnek üzerinde kullanımına bakalım. Örnekte, bir *SQL Server* veri tabanına bağlanmayı, belirli bir sorguyu çalıştırmayı, sorgu sonuçlarını bir JSON dizesine dönüştürmeyi ve ardından bu

JSON dizesini tekrar bir *TCLJSONQuery* nesnesine dönüştürerek ekrana mesaj olarak yazdırmayı amaçlamaktadır.

```
var
  kaynakSorgu: TClSqlQuery;
  jsonString: String;
  qry:TCLJSONQuery;
{
  kaynakSorgu = TClSqlQuery.Create(nil);

  Clomosy.DBSQLServerConnect('SQL Server', 'PC1', 'pc1', 'p**2', 'Urunler', 1433);

  try
    kaynakSorgu.Connection = Clomosy.DBSQLServerConnection;

    kaynakSorgu.SQL.Text = 'SELECT urunAdi, cihazTipi, urunFiyati FROM UrunBilgileri
WHERE model = ' + QuotedStr('Alpha');

    kaynakSorgu.Open;

    jsonString = Clomosy.DBDatasetGetAsJSON(kaynakSorgu);
    qry = Clomosy.ClDataSetFromJSON(jsonString);

    ShowMessage(qry.getJSonString);
  finally
    kaynakSorgu.Close;
    kaynakSorgu.Free;
  }
}
```

getJSONString

getJSONString özelliği, *TCLJSONQuery* türündeki bir nesnenin içindeki veriyi json formatında bir dizeye dönüştürerek döndüren bir özelliktir.

```
var
  qry:TCLJSONQuery;
{
   try
      qry = TCLJSONQuery.Create(nil);
      qry=
Clomosy.ClDataSetFromJSON('[{"ad":"Arda","yas":20},{"ad":"Ayşe","yas":27}]');
      ShowMessage(qry.getJSONString);
   except

     ShowMessage('Exception Class: '+LastExceptionClassName+' Exception Message:
'+LastExceptionMessage);
   }
}
```

RecordCount

RecordCount özelliği, bir json dizisinde bulunan nesne sayısını sayan bir işlevdir.

Bu işlev, bir json dizisinin metin temsilini alır ve bu dizide kaç tane nesne olduğunu sayar. Tipik olarak, bu tür bir işlev, bir json dizisinin içindeki öğelerin sayısını öğrenmek istediğiniz durumlarda kullanılır.

Aşağıdaki örnekte oluşturulmuş json verisi setinde kaç adet kaydolduğunu döndüren bir program bulunmaktadır. Sonuç olarak 2 değerini döndürmektedir.

```
var
  qry:TCLJSONQuery;
  kayitAdedi : Integer;
{

   try
     qry = TCLJSONQuery.Create(nil);
     qry =
Clomosy.ClDataSetFromJSON('[{"ad":"Arda","yas":20},{"ad":"Ayşe","yas":27}]');
       if (qry.Found)
       {
         kayitAdedi = qry.RecordCount;
         ShowMessage(kayitAdedi);
       }
   Except

     ShowMessage('Exception Class: '+LastExceptionClassName+' Exception Message:
'+LastExceptionMessage);
   }
}
```

14.8. Dosya İşlemleri

Dosya işlemleri sırasında projenin dosya yoluna erişim sağlamak için kullanılan özellikler mevcuttur.

AppBasePath

Clomosy uygulamasının başlatıldığı ana dizinin yolunu temsil eden bir özelliktir. Özellik string veri türü döndürmektedir.

```
var
  uygulamaYolu : String;

{
  uygulamaYolu = Clomosy.AppBasePath;
  ShowMessage('Uygulamanın ana dizin yolu: '+uygulamaYolu);
}
```

Örnek çıktı; D:\clomosy\

AppFilesPath

Uygulamanın dosya yolunu temsil eden bir özelliktir.

```
var
  uygulamaDosyaYolu : String;

{
  uygulamaDosyaYolu = Clomosy.AppFilesPath;
  ShowMessage('Uygulamanın dosya yolu: '+uygulamaDosyaYolu);
}
```

Örnek çıktı; D:\clomosy\Temporary\17020EA2217\02EE3ABE405\

14.9. Sistem Erişimi Özellikleri

ProcessMessages

ProcessMessages fonksiyonu, bir uygulamanın mesaj kuyruğundaki bekleyen olayları (events) işleyerek, arayüz güncellemelerini gerçekleştirmesine yardımcı olan bir Clomosy fonksiyonudur.

Clomosy uygulamaları, bir kullanıcının etkileşimleri, sistem olayları veya zamanlayıcı gibi çeşitli olaylara tepki olarak olaylar üretir. Bu olaylar, uygulama içindeki mesaj kuyruğuna eklenir. *ProcessMessages* fonksiyonu, bu mesaj kuyruğunu işleyerek, bekleyen olayları gerçekleştirir.

```
Clomosy.ProcessMessages;
```

setClipBoard

Metin veya verileri panoya kopyalamak için kullanılır. Bu fonksiyon bir parametre almaktadır. Parametre veri türü variant'tır.

Aşağıda örnekte bir butona tıklandığında prosedür tetiklenir ve panoya kopyalama işlemi yapılır.

```
var
 anaForm:TclForm;
 kopyalamaBtn : TclButton;
 x : Float;

void kopyalamaIslemi;
{
 x = Random() * 10;
 Clomosy.setClipBoard(Round(x));
 ShowMessage('Kopyalandı!');
}
{
 anaForm = TclForm.Create(Self);
```

```
kopyalamaBtn= anaForm.AddNewButton(anaForm,'kopyalamaBtn','Panoya Kopyala!');
kopyalamaBtn.TextSettings.Font.Size=50;
kopyalamaBtn.Align = alCenter;
kopyalamaBtn.Height = 50;
kopyalamaBtn.Width = 200;
anaForm.AddNewEvent(kopyalamaBtn,tbeOnClick,'kopyalamaIslemi');

anaForm.Run;
}
```

GetClipBoard

Panoya kopyalanan bir veri var ise bunun alınmasını sağlar.

Örnek olarak yukarıda gösterlmiş olan örneği kullanalım. Butona tıklandığında kopyalanan veri, butonun caption özelliğine atanacak.

```
void kopyalamaIslemi;
{
 x = Random() * 10;
Clomosy.setClipBoard(Round(x));
 ShowMessage('Kopyalandı!');
 kopyalamaBtn.Caption = Clomosy.GetClipBoard;
}
```

GetCurrentLocation

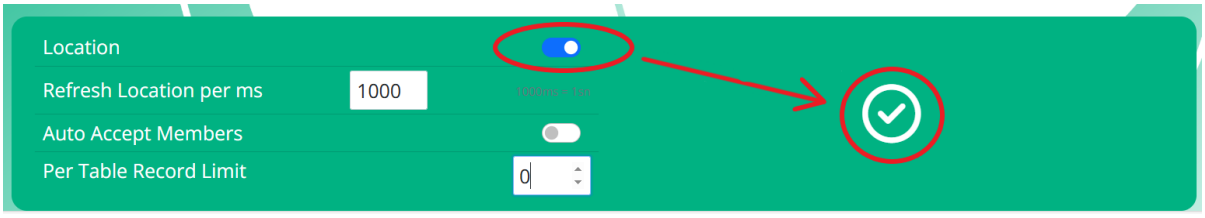
Mobil uygulamalarda cihazın mevcut konumunu elde etmek için bir fonksiyon kullanılmaktadır. Bu işlev, cihazın coğrafi koordinatlarını (enlem ve boylam) döndürmek için genellikle GPS veya ağ tabanlı konum hizmetlerini kullanır.

Konum verilerine erişimi etkinleştirmenin iki yöntemi vardır:

1) Uygulamada kullanıcının konum verilerine erişmeden önce ***GetCurrentLocation*** kullanılarak cihazın konum verileri elde edilmektedir.

```
Clomosy.GetCurrentLocation;
```

2) Web geliştirme ortamında Yönetim Paneli (Management)'ne gidin ve ardından Parametreler (Params)'e tıklayın. Açılan sayfada Konum ayarının etkinleştirilmesi gerekmektedir.



Daha sonra enlem ve boylam verilerini elde etmek için ***LocationValue*** özelliği kullanılmalıdır. *LocationValue* özelliği, Latitude|Longitude biçiminde bir dize döndürür.

Aşağıdaki örnekte konum verilerine erişim için ikinci yöntem kullanılmıştır. Ancak yorumda *GetCurrentLocation* özelliğinden bahsedildi. Daha sonra alınan enlem ve boylam bilgileri ayrıştırılarak ekranda görüntülenir.

```
var
  form1:TCLForm;
  btn1 : TclButton;
  Latitude,Longitude : String;

 void onBtnClick;
 {
  ShowMessage(Clomosy.LocationValue); //// Konum verileri erişimi => Latitude|Longitude
  Latitude=clGetStringTo(Clomosy.LocationValue,'|')
  Longitude=clGetStringAfter(Clomosy.LocationValue,'|')

  ShowMessage('Latitude value: '+Latitude);
  ShowMessage('Longitude value: '+Longitude);
 }
 {
  form1 = TCLForm.Create(Self);
  //Clomosy.GetCurrentLocation;
  btn1 = form1.AddNewButton(form1,'btn1','Erişim verilerini al');
  btn1.Width = 150;
  form1.AddNewEvent(btn1,tbeOnClick,'onBtnClick');
  form1.Run;

 }
```

ClearTemporary

ClearTemporary özelliği, Clomosy Learn uygulamasında kullanılan projelerin geçici dosyalarının yanı sıra kullanıcının proje bazında oluşturduğu dosyaların temizlenmesi için kullanılır.

Örnekte bir butona tıklandığına girilen projeye ait oluşturulan dosya içerisinde kayıtlı olan bütün dosyaları silecektir.

```
var
   form1:TCLForm;
   btn1 : TclButton;
void onBtnClick;
{
 //ShowMessage(Clomosy.AppFilesPath);
 Clomosy.ClearTemporary;
 ShowMessage('Dosyalar temizlendi.');
```

```
}
{
  form1 = TCLForm.Create(Self);

  btn1 = form1.AddNewButton(form1,'btn1','Proje dosyasını temizle!');
  btn1.Width = 150;
  form1.AddNewEvent(btn1,tbeOnClick,'onBtnClick');

  form1.Run;
}
```

14.10. Dosya ve Base64 Kodlama/Kod Çözme Fonksiyonları

StreamToBase64

Clomosy'de StreamToBase64, bir veri akışını (TCLMemoryStream) Base64 formatına dönüştürmek için kullanılan bir işlevdir. Base64, ikili verileri metin formatına dönüştürmek için kullanılan bir kodlama şemasıdır. Bu yöntem, verilerin e-posta veya JSON gibi metin tabanlı protokoller aracılığıyla güvenli bir şekilde iletilmesine veya saklanmasına olanak tanır.

StreamToBase64 fonksiyonu bir parametre almaktadır. Bu parametre TclMemoryStream tipinde olmalıdır. Geriye string türünde bir veri döndürmektedir.

```
Clomosy.StreamToBase64(AStream:TclMemoryStream):string;
```

Aşağıda TclMemoryStream nesnesini Base64 formatına dönüştüren örnek bir kod parçacığı verilmiştir. Örnekte bir dosyadan elde edilen görüntü TclMemoryStream'e dönüştürülmekte ve dönüştürülen bu yapı daha sonra Base64 formatına dönüştürülerek bir string nesnesinde saklanmaktadır.

```
var
  Form1 : TCLForm;
  btnImgYukle : TClProButton;
  Img1 : TCLImage;
  memoryStream : TCLMemoryStream;
  base64String,dosyaAdiKaydet : String;

 void ConvertMemoryStreamToBase64
 {
  if (memoryStream.Size > 0) // memoryStream içeriğinin boş olup olmama durumu kontrolü
  {
   base64String = Clomosy.StreamToBase64(memoryStream); // TclMemoryStream'i Base64'e dönüştür
   //dosyaAdiKaydet = clPathCombine('apple.png', Clomosy.AppFilesPath);
   //Clomosy.Base64ToFile(dosyaAdiKaydet,base64String); // Verileri belirtilen uzantıyla base64 olarak kaydeder.
  }
```

```
  else  ShowMessage('MemoryStream Boş');
}

void loadImageButtonClick
var
  MyFileStr : String;
{
  MyFileStr = clPathCombine('apple.png', Clomosy.AppFilesPath);
  if clFileExists('apple.png', Clomosy.AppFilesPath)
  {
    memoryStream = Clomosy.FileToStream(MyFileStr);
    Img1.Bitmap.LoadFromStream(memoryStream);
    //Img1.Bitmap.LoadFromFile(MyFileStr);
    ShowMessage('Resim Yüklendi.');
    ConvertMemoryStreamToBase64;
  }
  else
    ShowMessage('Dosya Yok');
}

{
  Form1 = TCLForm.Create(Self);
  Form1.AddAssetFromUrl('https://clomosy.com/demos/apple.png');
  MemoryStream = TclMemoryStream.Create;

  Img1 = Form1.AddNewImage(Form1,'Img1');
  Img1.Width = 100;
  Img1.Height = 100;

  btnImgYukle = Form1.AddNewProButton(Form1,'btnImgYukle','Resim Yükle');
  btnImgYukle.Align = alMostBottom;
  btnImgYukle.Margins.Top = 20;
  btnImgYukle.clProSettings.FontSize = 16;
  btnImgYukle.clProSettings.FontColor = clAlphaColor.clHexToColor('#ffffff');
  btnImgYukle.clProSettings.BackgroundColor = clAlphaColor.clHexToColor('#6966ff');
  btnImgYukle.clProSettings.RoundHeight = 10;
  btnImgYukle.clProSettings.RoundHeight = 10;
  btnImgYukle.SetclProSettings(btnImgYukle.clProSettings);
  Form1.AddNewEvent(btnImgYukle,tbeOnClick,'loadImageButtonClick');
  Form1.Run;
}
```

FileToStream

FileToStream, bir dosyanın içeriğini bir akışa, özellikle de bir *TclMemoryStream'e* yükleme işlemini ifade eder. Bu, bir dosyanın içeriğinin okunmasını ve bunların bellekteki bir *TclMemoryStream* örneğine aktarılmasını içerir. Bu yöntem, dosya içeriklerinin bellekte veya başka bir veri kaynağında temsil edilmesini sağlayarak içerik üzerinde daha esnek işlemler yapılmasına olanak tanır.

FileToStream fonksiyonu bir parametre almaktadır. Bu parametre görselin dosya yolu olmalıdır. Geriye *TclMemoryStream* türünde bir veri döndürmektedir.

```
Clomosy.FileToStream(AFileName:string):TclMemoryStream;
```

Aşağıdaki örnekte uygulama başlatıldığında *AddAssetFromUrl* fonksiyonu kullanılarak proje dosyalarına bir görsel eklenmektedir. Daha sonra bir butona tıklandığında bu görüntü, bir *TclMemoryStream* döndüren *FileToStream* işlevi kullanılarak bir *TclMemoryStream* nesnesine dönüştürülür. Bu nesne daha sonra bir görüntü bileşenine (Img1) atanır.

```
var
  Form1 : TCLForm;
  btnImgYukle : TClProButton;
  Img1 : TCLImage;
  memoryStream : TCLMemoryStream;

 void resimyukleBtnEvent
 var
  MyFileStr : String;
 {
  MyFileStr = clPathCombine('apple.png', Clomosy.AppFilesPath);
  if clFileExists('apple.png', Clomosy.AppFilesPath)
  {
   memoryStream = Clomosy.FileToStream(MyFileStr);
   Img1.Bitmap.LoadFromStream(memoryStream);
   //Img1.Bitmap.LoadFromFile(MyFileStr);
   ShowMessage('Resim yüklendi.');
  }
  else
   ShowMessage('Dosya yok');
 }

 //------ Ana Kod -------------
 {
  Form1 = TCLForm.Create(Self);
  Form1.AddAssetFromUrl('https://clomosy.com/demos/apple.png');
  MemoryStream = TclMemoryStream.Create;

  Img1 = Form1.AddNewImage(Form1,'Img1');
  Img1.Width = 100;
  Img1.Height = 100;

  btnImgYukle = Form1.AddNewProButton(Form1,'btnImgYukle','Resim Yükle');
  btnImgYukle.Align = alMostBottom;
  btnImgYukle.Margins.Top = 20;
  btnImgYukle.clProSettings.FontSize = 16;
  btnImgYukle.clProSettings.FontColor = clAlphaColor.clHexToColor('#ffffff');
  btnImgYukle.clProSettings.BackgroundColor = clAlphaColor.clHexToColor('#6966ff');
```

```
  btnImgYukle.clProSettings.RoundHeight = 10;
  btnImgYukle.clProSettings.RoundHeight = 10;
  btnImgYukle.SetclProSettings(btnImgYukle.clProSettings);
  Form1.AddNewEvent(btnImgYukle,tbeOnClick,'resimyukleBtnEvent');

  Form1.Run;
 }
```

FileToBase64

FileToBase64, Clomosy'de bir dosyanın içeriğini Base64 formatına dönüştürmek için kullanılan bir işlevdir. Bu işlev bir dosyanın içeriğini okur, onu bellekteki bir *TclMemoryStream* nesnesine yükler ve ardından bu akışı Base64 biçimine dönüştürür.

FileToBase64 fonksiyonu bir parametre almaktadır. Bu parametre görselin dosya yolu olmalıdır. Geriye string türünde bir veri döndürmektedir.

```
Clomosy.FileToBase64(AFileName:string):string;
```

Örnekte bir butona tıklandığında proje dosyalarından elde edilen görseli Base64 formatında bir *TclMemo* nesnesine yazmaktadır.

```
var
  Form1 : TCLForm;
  btnResimYukle : TClProButton;
  memoNot : TclMemo;

 void memoFileToBase64Click;
 {
  memoNot.Lines.Text =
Clomosy.FileToBase64(clPathCombine('apple.png',Clomosy.AppFilesPath));
 }

 {
  Form1 = TCLForm.Create(Self);
  Form1.AddAssetFromUrl('https://clomosy.com/demos/apple.png');

  memoNot = Form1.AddNewMemo(Form1,'memoNot','---');
  memoNot.Align = alMostTop;
  memoNot.Height = 300;
  memoNot.StyledSettings = ssFamily;
  memoNot.TextSettings.WordWrap = True;

  btnResimYukle = Form1.AddNewProButton(Form1,'btnResimYukle','Add
FileToBase64');
  btnResimYukle.Align = alMostBottom;
  btnResimYukle.Margins.Top = 20;
  btnResimYukle.clProSettings.FontSize = 16;
```

```
  btnResimYukle.clProSettings.FontColor = clAlphaColor.clHexToColor('#ffffff');
  btnResimYukle.clProSettings.BackgroundColor =
clAlphaColor.clHexToColor('#6966ff');
  btnResimYukle.clProSettings.RoundHeight = 10;
  btnResimYukle.clProSettings.RoundHeight = 10;
  btnResimYukle.SetclProSettings(btnResimYukle.clProSettings);
  Form1.AddNewEvent(btnResimYukle,tbeOnClick,'memoFileToBase64Click');

  Form1.Run;
 }
```

Base64ToFile

Base64ToFile, Clomosy'de Base64 dizesini bir dosyaya dönüştürmek için kullanılan bir işlevdir. Bu işlev, yeni bir dosya oluşturarak veya mevcut bir dosyanın içeriğinin üzerine yazarak Base64 dizesinin istenen dosya yoluna yazılmasına olanak tanır.

Base64ToFile fonksiyonu iki parametre almaktadır. İlk parametre dosya yolu, ikinci parametre ise Base64 verisini içermektedir.

```
Clomosy.Base64ToFile(AFileName,ABase64:string);
```

Örnekte, proje dosyası dizininde bulunan bir görüntü Base64 formatına dönüştürülerek bir *TclMemo* nesnesine yazılmaktadır. Daha sonra Base64 formatındaki görüntüyü içeren bu nesne, *Base64ToFile* özelliği kullanılarak yeni bir görüntünün istenilen proje dizinine kaydedilmesi için kullanılır.

```
var
  Form1 : TCLForm;
  btnYukleImg : TClProButton;
  memoNot : TclMemo;

 void memoFileToBase64Click;
 {
  memoNot.Lines.Text =
Clomosy.FileToBase64(clPathCombine('apple.png',Clomosy.AppFilesPath));

Clomosy.Base64ToFile(clPathCombine('newApple.png',Clomosy.AppFilesPath),memoNot
.Lines.Text);
 }
 {
  Form1 = TCLForm.Create(Self);
  Form1.AddAssetFromUrl('https://clomosy.com/demos/apple.png');

  memoNot = Form1.AddNewMemo(Form1,'memoNot','---');
  memoNot.Align = alMostTop;
  memoNot.Height = 300;
  memoNot.StyledSettings = ssFamily;
```

```
  memoNot.TextSettings.WordWrap = True;

  btnYukleImg = Form1.AddNewProButton(Form1,'btnYukleImg','FileToBase64 Ekle');
  btnYukleImg.Align = alMostBottom;
  btnYukleImg.Margins.Top = 20;
  btnYukleImg.clProSettings.FontSize = 16;
  btnYukleImg.clProSettings.FontColor = clAlphaColor.clHexToColor('#ffffff');
  btnYukleImg.clProSettings.BackgroundColor = clAlphaColor.clHexToColor('#6966ff');
  btnYukleImg.clProSettings.RoundHeight = 10;
  btnYukleImg.clProSettings.RoundHeight = 10;
  btnYukleImg.SetclProSettings(btnYukleImg.clProSettings);
  Form1.AddNewEvent(btnYukleImg,tbeOnClick,'memoFileToBase64Click');

  Form1.Run;
 }
```

Base64ToStream

Base64ToStream, Base64 dizesini bir akışa (*TCLMemoryStream*) dönüştürmek için kullanılan bir işlevdir. Bu işlev bir Base64 kodunu alır ve onu bir *TCLMemoryStream* nesnesine dönüştürerek verileri okunabilir hale getirir.

Base64ToStream fonksiyonu bir parametre almaktadır. Bu parametre Base64 verisini içermektedir. Geriye TclMemoryStream nesnesine ait veri döndürmektedir.

```
Clomosy.Base64ToStream(ABase64:string):TclMemoryStream;
```

Örnekte, *TclMemo* nesnesi içinde Base64 formatında tutulan veriler, *Base64ToStream* işlevi kullanılarak bir *TCLMemoryStream* nesnesine dönüştürülür. Dönüştürülen bu veriler daha sonra bir *TclImage* nesnesine aktarılır.

```
var
  Form1 : TCLForm;
  memoNot : TclMemo;
  imgKaynak : TCLImage;

 void memoBase64ToStreamClick;
 var
 LMemStream:TCLMemoryStream;
 {
  memoNot.Lines.Text =
Clomosy.FileToBase64(clPathCombine('apple.png',Clomosy.AppFilesPath));
  LMemStream = Clomosy.Base64ToStream(memoNot.Lines.Text);
  imgKaynak.Bitmap.LoadFromStream(LMemStream);
 }

 {
  Form1 = TCLForm.Create(Self);
```

```
  Form1.AddAssetFromUrl('https://clomosy.com/demos/apple.png');

  memoNot = Form1.AddNewMemo(Form1,'memoNot','---');
  memoNot.Align = alMostTop;
  memoNot.Height = 400;
  memoNot.StyledSettings = ssFamily;
  memoNot.TextSettings.WordWrap = True;
  Form1.AddNewEvent(memoNot,tbeOnClick,'memoBase64ToStreamClick');
  imgKaynak = Form1.AddNewImage(Form1,'imgKaynak');
  imgKaynak.Align = alClient;
  Form1.setImage(imgKaynak,'https://clomosy.com/learn/xMark.png');

  Form1.Run;
 }
```

14.11. Kripto Şifreleme Algoritması

Kripto şifreleme algoritmalarının temel amacı, bilgi güvenliğini sağlamak ve hassas verilerin yetkisiz erişimden korunmasına yardımcı olmaktır. Bu algoritmalar, şifreleme ve şifre çözme süreçleri kullanarak verilerin gizliliğini sağlar.

Bilgiyi güvenli bir şekilde iletmek ve depolamak için kripto şifreleme kullanılır. Bu sayede, veriler yetkisiz erişimden korunarak sadece doğru anahtara sahip olan kişiler tarafından okunabilir.

Kripto şifreleme algoritmaları, verilerin değiştirilmediğini veya bozulmadığını doğrulamak için kullanılır. Veri bütünlüğü, verilerin hedeflenmeyen değişikliklere karşı korunmasını sağlar.

Clomosy platformu üzerinde kripto şifreleme algoritması olarak AES şifreleme kullanılır. AES, "Advanced Encryption Standard"ın (Gelişmiş Şifreleme Standardı) kısaltmasıdır. Bu algoritma, simetrik şifreleme için tasarlanmış bir blok şifreleme algoritmasıdır. Simetrik şifreleme, aynı anahtarın hem şifreleme hem de şifre çözme işlemlerinde kullanıldığı bir şifreleme türüdür.

Clomosy de bir metni şifreleme işlemi için *ProjectEncryptAES* fonksiyonu kullanılır. Bu fonksiyon string veri türünde parametre almaktadır. Şifrelenen metni çözebilmek için ise *ProjectDecryptAES* fonksiyonu kullanılır. Bu fonksiyon da string veri türünde parametre almaktadır.

Örnek kullanımına bakalım. Örnekte bir metin şifreleniyor mesaj kutusunda gösteriliyor. Ardından şifrelenen metin çözülüyor ve yine çözümlenmiş hali gösteriliyor.

```
var
  metin: string;
  sifrelenmisDeger: string;
  cozulenDeger: string;
{
  metin = 'Clomosy Learn';
```

```
  sifrelenmisDeger = Clomosy.ProjectEncryptAES(metin);
  ShowMessage('Şifrelenmiş Değer: ' + sifrelenmisDeger);

  cozulenDeger = Clomosy.ProjectDecryptAES(sifrelenmisDeger);
  ShowMessage('Çözülmüş Değer: ' + cozulenDeger);
}
```

14.12. Program Durdurma (sleepAndCall)

Clomosy programlama dilinde *sleepAndCall* fonksiyonu, programın belirli bir süre boyunca duraklamasını sağlayan bir fonksiyondur. Bu fonksiyon, belirli bir süre boyunca programın işlemlerini durdurarak, belirtilen süre kadar bekletir.

sleepAndCall fonksiyonu belirli bir süre bekleyen ve ardından belirtilen bir geri çağırma işlevini yürüten bir fonksiyonu temsil eder. Bu tür bir işlev, belirli bir görevin tamamlanmasını beklemek ve ardından başka bir işlevi yürütmeye devam etmek için kullanılabilir.

Fonksiyon üç parametre almaktadır. İlk parametre beklenecek süreyi (milisaniye cinsinden) belirtir. İkinci parametre bekleme sürecinden önce tetiklenecek eylemi temsil eder. Üçüncü parametre, bekleme işlemi tamamlandıktan sonra belirtilen geri çağırma fonksiyonunu çalıştırır.

```
Clomosy.SleepAndCall(3000,'durdurmadanOncekiFonk','durdurmadanSonrakiFonk');
```

Aşağıdaki örnekte bir butona tıklandığında *sleepAndCall* fonksiyonu çağrılır. Ardından buton başlığı fonksiyondan öncesi ve sonrası olarak değiştirilir.

```
var
  anaForm:TclForm;
  durdurmaBtn : TclButton;

void durdurmadanSonra;
{
  durdurmaBtn.Text = 'Beklet Sonrası';
}

void durdurmadanOnce;
{
  durdurmaBtn.Text = 'Beklet Öncesi';
}
void sleep
{
  Clomosy.SleepAndCall(3000,'durdurmadanOnce','durdurmadanSonra');
}
{
  anaForm = TclForm.Create(Self);
```

```
  durdurmaBtn= anaForm.AddNewButton(anaForm,'durdurmaBtn','Beklet!');
  durdurmaBtn.TextSettings.Font.Size=30;
  durdurmaBtn.Align = alCenter;
  durdurmaBtn.Height = 70;
  durdurmaBtn.Width = 200;
  anaForm.AddNewEvent(durdurmaBtn,tbeOnClick,'sleep');
  anaForm.Run;
}
```

14.13. Diğer Erişim Özellikleri

14.13.1. Global Değişken Tanımlaması

Global değişkenler, bir programın herhangi bir bölümünde tanımlanmış ve kullanılabilecek değişkenlerdir. Bu değişkenler genellikle programın ana bölümünde, yani tüm fonksiyonların ve prosedürlerin erişebileceği bir alanda tanımlanır. Global değişkenler, genellikle programın durumunu veya genel ayarlarını saklamak için kullanılır.

Clomosy platformunun sunmuş olduğu global değişken tanımlama fonksiyonları vardır. Bu fonksiyonlar farklı birimler üzerinde ortak kullanılacak değerleri saklamak için tasarlanmıştır.

Örneğin *Ana Kod* (*Main Code*) sayfasında tanımlanan bir string değişken bir sonraki sayfaya (*Unit*) göndermek ve o sayfada kullanılmak istendiğinde bu fonksiyonlar ile tanımlanabilir.

GlobalVariableString

String veri tipinde saklanacak verilerin tutulduğu fonksiyondur.

```
Clomosy.GlobalVariableString = 'Test';
```

Bir örnek üzerinde kullanalım.

Örnekte ana kod ekranında bir meyve seçildiğini düşünelim. Bu bilgi global değişkende tutularak bir sonraki sayfaya geçilir ve o sayfada gelen veri mesaj kutusunda gösterilir.

Ana Kod:

```
var
  meyve : String;
{
  meyve = 'Elma';
  Clomosy.GlobalVariableString = meyve;
  Clomosy.RunUnit('uMeyve');
}
```

Unit (Uygulamada uMeyve olarak oluşturuldu):

```
var
  secilenMeyve : String;
{
  secilenMeyve = Clomosy.GlobalVariableString;
  ShowMessage('Seçilen meyve: '+secilenMeyve);
}
```

GlobalVariableInteger

Integer veri tipinde saklanacak verilerin tutulduğu fonksiyondur.

```
Clomosy.GlobalVariableInteger = 20;
```

Örnekte rastgele üretilmiş bir sayıyı global veri değişkenine atılarak bir sonraki sayfada gösterilir.

Ana Kod:

```
var
  sayi : Integer;
{
  sayi = Trunc(Random()*20);
  Clomosy.GlobalVariableInteger = sayi;
  Clomosy.RunUnit('uRandmSayi');
}
```

Unit (Uygulamada uRandmSayi olarak oluşturuldu):

```
var
  randmSayi : Integer;
{
  randmSayi = Clomosy.GlobalVariableInteger;
  ShowMessage('Gelen sayi: '+IntToStr(randmSayi));
}
```

GlobalVariableDateTime

DateTime veri tipinde saklanacak verilerin tutulduğu fonksiyondur.

```
Clomosy.GlobalVariableDateTime= 28.01.2024 00:19:43;
```

Örnekte bugünün tarihi global veri değişkenine atılarak bir sonraki sayfada gösterilir.

Ana Kod:

```
var
  tarih : TclDateTime;
{
  tarih = Now;
  Clomosy.GlobalVariableDateTime = tarih;
  Clomosy.RunUnit('uTarih');
}
```

Unit (Uygulamada uTarih olarak oluşturuldu):

```
var
  gelenTarihVerisi : Integer;
{
  gelenTarihVerisi = Clomosy.GlobalVariableDateTime;
  ShowMessage('Tarih Bilgisi: '+DateTimeToStr(gelenTarihVerisi));
}
```

GlobalVariableStringList

Bir TclStringList veri tipindeki diziyi depolamak ve saklamak için tutulan fonksiyondur. Örnek üzerinde kullanalım. Ana kod alanında TclStringList nesnesi oluşturulur. Birkaç ürün eklenir ve bu nesne oluşturulan unit'e yönlendirilir. Bu unit (uUrunler) ile gelen liste içindeki ürünler döngü ile gösterilir.

Ana Kod:

```
var
  urunListesi: TclStringList;
  i: Integer;
{
  urunListesi = Clomosy.StringListNew;
  urunListesi.Add('Ürün 1');
  urunListesi.Add('Ürün 2');
  urunListesi.Add('Ürün 3');

  Clomosy.GlobalVariableStringList = urunListesi;

  Clomosy.RunUnit('uUrunler');
}
```

Unit (Uygulamada uUrunler olarak oluşturuldu):

```
var
  gelenUrunListesi: TclStringList;
  i: Integer;
{
  gelenUrunListesi = Clomosy.StringListNew;
  gelenUrunListesi = Clomosy.GlobalVariableStringList;

  for (i = 0 to gelenUrunListesi.count - 1)
    ShowMessage(Clomosy.StringListItemString(gelenUrunListesi,i));
}
```

GlobalBitmapSaveToFile

Bir görüntüyü veya bitmap'i bir dosyaya dönüştürmeye ve istenen dizine kaydetmeye olanak tanır. Bu fonksiyon bir parametre almaktadır. Parametre içerisine string tipinde dosyanın kaydedileceği konumu ve dosya adının girilmesi gerekmektedir. Aşağıda bir örnek tanımlama mevcuttur.

```
Clomosy.GlobalBitmapSaveToFile('D:\clomosy\GlobalBitmap.png');
```

Bir örnek ile kullanımına bakalım. Örnekte bir QR kod üreticisi bulunmaktadır. Uygulama her 10 saniyede bir tarih ve saati içerisinde barındıran bir QR kod üretir. Üretildiği anda istenilen dosya uzantısında resim olarak saklanır. QR kod her değiştiğinde resim dosyası da güncellenmektedir.

```
var
  anaForm:TclForm;
  QRGen : TClQRCodeGenerator;
  qrZamanlayici:TClTimer;
  qrKodBaslikLbl:TclLabel;

void resimKaydet;
{
  Clomosy.GlobalBitmapSaveToFile('D:\clomosy\GlobalTBitmap.png');
}

void QrKodUret;
{
  QRGen.Text = FormatDateTime('ddmmyy hhnnss', Now);
  qrKodBaslikLbl.Caption = QRGen.Text;
}

void qrkodZamanlayiciBaslat;
{
  QrKodUret;
}
```

```
{
  anaForm = TclForm.Create(Self);

  qrKodBaslikLbl= anaForm.AddNewLabel(anaForm,'qrKodBaslikLbl','QrKod Bilgisi');
  qrKodBaslikLbl.Align = alTop;

  QRGen= anaForm.AddNewQRCodeGenerator(anaForm,'QRGen','Hello World');
  QRGen.Height = 200;
  QRGen.Align = alCenter;
  anaForm.AddNewEvent(QRGen,tbeOnGetQRCode,'resimKaydet');

  qrZamanlayici= anaForm.AddNewTimer(anaForm,'qrZamanlayici',10000);
  qrZamanlayici.Enabled = True;
  anaForm.AddNewEvent(qrZamanlayici,tbeOnTimer,'qrkodZamanlayiciBaslat');

  anaForm.Run;
}
```

14.13.2. Kamera Erişimi

ImageChooser

Proje kapsamındaki kullanıcıların galeriden veya dosya sistemlerinden bir görüntü seçip, kameradan fotoğraf çekmesine olanak tanımaktadır. Bu özellik iki parametre almaktadır. İlk parametre görüntü nesnesinin konumlanacağı formu ve bileşenleri temsil etmektedir. İkinci parametre ise çekilen veya seçilen fotoğrafın yerleştirileceği görüntü nesnesini belirtmektedir.

Kullanım:

```
Clomosy.ImageChooser(TclForm, TclImage);
```

Örnek ile kullanalım. Bir buton oluşturulsun ve butona tıklandığın kamera açılsın ardından fotoğraf çekme veya galeriden resim seçme özelliği açılarak işlem yapılır. Ardından bir TclImage nesnesine görsel aktarılır.

```
var
  anaForm:TclForm;
  kameraBtn:TclButton;
  kameraImg:TClImage;

void onbtnClick;
{
  Clomosy.ImageChooser(anaForm, kameraImg);
}
{
  anaForm = TClForm.Create(Self);
  kameraBtn = anaForm.AddNewButton(anaForm, 'kameraBtn','Kamerayı aç!');
  anaForm.AddNewEvent(kameraBtn, tbeOnClick, 'onbtnClick');
  kameraBtn.Align = alBottom;
```

```
  kameraImg = anaForm.AddNewImage(anaForm, 'kameraImg');

  anaForm.setImage(kameraImg,'https://clomosy.com/demos/balloon.png');
  kameraImg.Align = alCenter;
  kameraImg.Height = 300;
  kameraImg.Width = 300;

  anaForm.Run;
}
```

14.13.3. Uygulama Çalışma Süreci Kaydı

EventLog

Log, bir uygulamanın çalışma sürecinde ortaya çıkan olayları, hataları veya önemli bilgileri kaydetmek amacıyla kullanılan bir kayıt türüdür. Loglar, genellikle uygulamanın durumunu izlemek, hata ayıklama yapmak veya performans analizi yapmak için kullanılmaktadır.

Clomosy.EventLog özelliği, Clomosy platformunda log tutma işlemini etkinleştirmek veya devre dışı bırakmak için kullanılmaktadır. *True* değeriyle ayarlandığında, uygulama belirli olayları kaydetmeye başlar ve bu loglar daha sonra inceleme veya hata ayıklama süreçlerinde kullanılabilir. Eğer *False* (Yanlış) olarak ayarlanırsa, loglama devre dışı bırakılır.

Loglar genellikle bir dosyada veya başka bir depolama mekanizmasında saklanır ve uygulamanın çalışması sırasında meydana gelen olayların bir izini bırakmak için kullanılmaktadır. Bu izleme işlemi, uygulamanın performansını analiz etmek, hataları tespit etmek ve kullanıcı deneyimini iyileştirmek için önemlidir.

Kullanımı:

```
Clomosy.EventLog = True;
```

BÖLÜM 15. Veri Tabanı Giriş ve Temel Veri Tabanı Kavramları

Veri tabanı, verileri derli toplu olarak saklamaya yarayan dosya ve dosyalar kümesidir. Temel olarak farklı tiplerdeki verileri düzenli bir şekilde saklamayı ve kullanmayı sağlamaktadır. Veri tabanı işlemleri ile verileri saklayabilir, onlara kolay bir şekilde ulaşabilir ve gerektiğinde bu verilerin üzerinde değişiklikler yapılmaktadır.

Veri tabanı, veriler arasında bütünlük ve düzen sağlarken, veriye hızlı erişim ve bakım kolaylığı da sağlamaktadır. Günümüzde veri tabanı, neredeyse her alanda kullanılmaktadır. Örneğin; adres defteri, telefon rehberi, sözcükler, hastane otomasyonları veya harita yazılımları veri tabanları ile haberleşerek sorgulama yapmaktadırlar.

15.1. Tablolar

Veri tabanında bilgilerin saklandığı alanlara tablo denir. Tablolar alanlar ve satırlardan oluşmaktadır. Bir veri tabanında pek çok bilgi saklanmaktadır. Ancak bunların belirli bir şekilde ayrılması gerekir. Örneğin bir okulun tüm öğrencilerinin ve tüm notlarının tek bir tablo içerisinde tutulması karmaşaya yol açabilir. Bu durumda veriler tablolara ayrılır ve tablolar birbirleriyle ilişkilendirilir. Bir veri tabanında aynı isimde iki tablo bulunamaz.

15.1.1. Tablo Alanları ve Veri Türleri

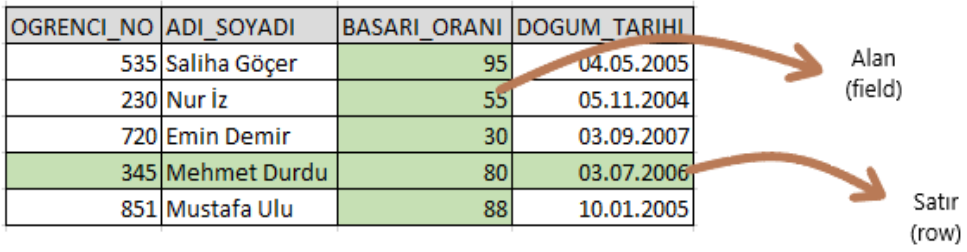
Veri tabanının tablolardan oluştuğunu belirtmiştik. Tablolar da alanlardan oluşmaktadır. Alanlar, aynı türden veri saklayan ve kayıtlara ait özellikleri barındıran birimlerdir. Örneğin, öğrenci bilgilerini saklayan bir tabloda, bir alan öğrenci numarasını, bir alan adını ve diğer bir alan da notunu tutar. Yani her bir özelleşmiş bilgi, bir alanda saklanır.

Her bir alanın, saklayacağı bilginin özelliğini belirten veri türü bulunmaktadır. Bu veri türü sayesinde o alanda saklanacak bilgi sınırlıdır. Örneğin sayısal ifadelerin yer alacağı bir alana karakter girilmesi veya tarihsel alana başka bir veri tipi girişi yapılması durumunda veri tabanı tarafından bir mesajla kullanıcılar uyarılır.

Satırlar (Row)

Bir tablodaki bilgiler satırlarda saklanmaktadır. Veri tabanı tablo yapılarındaki alan ve satır yapısı Excel yapısına benzemektedir. Excel sütunları, tabloların alanlarına, satırları ise tabloların satırlarına benzetilebilir. Tablolardaki her bir satır bir kayıt demektir.

Aşağıdaki şekilde, tablo yapısı üzerindeki alanlar ve satırlar gösterilmiştir.



Resim 67

15.2. Anahtarlar

Tablodaki kayıtları birbirinden ayırt edebilmek için tablo içindeki alanlara, belirli anahtarlar atayarak birçok işlem kolaylaşmaktadır. Bir tablo içerisinde, aşağıdaki anahtar yapıları tanımlanabilir.

Birincil Anahtar (primary key-PK)

Tekil Anahtar (unique key-UK)

Referans Anahtar (foreign key-FK)

Birleşik Anahtar (composite key-CK)

15.2.1. Birincil Anahtar

Bir tablo içindeki satırları birbirinden ayırt eden anahtar yapısıdır. Bir veya birden fazla alandan oluşmaktadır. Birincil anahtar olarak tanımlanan alan, boş değer (*null*) içermez ve içerdiği veri tekrarlanmaz. Örneğin öğrenci tablosunda, öğrenci numarasını tutan alan bu tablodaki tekil bilgiyi tutar. O zaman tasarım sırasında öğrenci numarasını tutan alan, tekil anahtar olarak belirtilmelidir.

15.2.2. Tekil Anahtar

Bir tablo da tekil anahtar olarak tanımlanmış bir alana aynı değer sadece bir kez girilmektedir. Birincil anahtardan farklı olarak, tabloda bu alana ait sadece bir kayıt **null (boş)** değeri almaktadır.

15.2.3. Referans Anahtar

Bir tablodaki alanı, başka bir tablonun belli bir alanında yer alabilecek veri grubu ile sınırlandırmaya ve ilişkilendirmeye yarayan anahtar yapısıdır. İlişkilendirilecek olan tablonun **Primary Key** alanı ile diğer tablonun **Foreign key** alanı birbiri ile bağlanmaktadır. Referans anahtar tanımlanırken, ilişkilendirilen tablolar arasında ne tür işlem ve kısıtlama yapılacağı belirtilmektedir.

15.2.4. Birleşik Anahtar

Birden fazla alanın birleştirilmesiyle, birincil anahtar görevini üstlenecek tanımlamalar yapılmaktadır. Bu şekilde tanımlanan anahtar yapıları, ‘birleşik anahtar’ olarak adlandırılmaktadır.

15.3. SQL

SQL (Structured Query Language), yapılandırılmış sorgu dilinin kısaltmasıdır. SQL, ilişkisel veri tabanı yönetim sistemlerinde (Relationships Database Management System-RDBMS) veri tabanlarını oluşturmak, sorgulamak, güncellemek ve yönetmek için kullanılan özel bir dildir.

Piyasada pek çok veri tabanı yönetim sistemi bulunmaktadır. Bunlardan en çok bilinenleri Oracle, Microsoft SQL Server, Interbase, Firebird, PostgreSQL, MySQL,Informix’tir.

Çeşitli programlama dilleri gibi, SQL de standart hale getirilmiştir. ANSI tarafından standart hale getirilmiş ANSI SQL, her bir veri tabanı yönetim sistemi tarafından desteklenmesi gereken bir standarttır.

SQL deyimleri, veri tabanları üzerinde çeşitli işlemleri yerine getirmektedir. Veri tabanından sorgulama yapmak için **SELECT**, ekleme yapmak için **INSERT** güncelleme yapmak için

UPDATE, silme için **DELETE**, yeni tablo oluşturmak için **CREATE TABLE** gibi komutlara sahiptir.

15.3.1. Select İfadesi (Kayıt Sorgulamak)

SQL'de *SELECT* ifadesi, bir veri tabanından veri çekmek için kullanılan temel sorgu ifadesidir. *SELECT* ifadesi, belirli bir tablodan veya birden fazla tablodan belirli sütunlardan veri seçilmesini sağlamaktadır. Ayrıca, bu ifade ile sorgular filtrelenir, sıralanır ve gruplandırılmaktadır.

Temel bir *SELECT* ifadesi şu şekildedir:

```
SELECT <AlanAdı1>, <AlanAdı2>, <AlanAdı3>, <AlanAdı4> FROM <TabloAdı>
```

Örnek kullanımı aşağıdaki gibidir:

```
SELECT OGRENCI_NO, AD, SOYAD, SINIF FROM OGRENCI
```

Yukarıdaki SQL cümlesinde tablo üzerinde bulunan tüm alanlar listelenmiştir. Tüm alanların tek tek yazılması yerine SELECT ile FROM arasına yıldız (*) sembolü koymak yeterli olacaktır.

```
SELECT * FROM OGRENCI
```

Her iki sorgu cümlesinde de öğrenci tablosunda bulunan bütün kayıtlar ve bütün alanlar birlikte listelenmektedir.

SQL ifadesinde yer alan sütunların yeri değiştirilebilir, listede yer alması istenmeyen sütunlar cümle içerisinden çıkarılabilir.

Yukarıdaki örnekte OGRENCI_NO, AD, SOYAD, SINIF alanları sıralı ve bütün alanlar listelenmişti. Şimdi sırasının değiştirilmesi ve sadece ad,soyad ve okul numarası verilerini listeleyelim.

```
SELECT AD, SOYAD, OGRENCI_NO FROM OGRENCI
```

Where İfadesi (Şart Belirtmek)

SQL sorgularına koşullar eklemek amacıyla kullanılan bir ifadedir. Veri tabanı tablolarında belli kayıtlar üzerinde sorgulama, güncelleme ve silme gibi önemli işlemleri gerçekleştirirken kullanılan bir SQL ifadesidir. Where ifadesi ile mantıksal ve karşılaştırma operatörleri kullanılarak, sadece istenen veri kümesi üzerinde işlem yapılır ve bu sayede daha hızlı ve performanslı işlemler gerçekleştirilir.

Karşılaştırma Operatörleri

Operatör	Açıklama
=	Eşittir
< >	Eşit değildir, farklıdır.
>	Büyüktür
<	Küçüktür
>=	Büyük veya eşittir
<=	Küçük veya eşittir
BETWEEN	Arasında
LIKE	Metin arama

Karşılaştırma operatörlerinde, karşılaştırdığımız veri tiplerine dikkat etmemiz gerekmektedir. Sayısal veriyi karakter tipinde bir veri ile karşılaştıramayız.

Karşılaştırma operatörleri 'Where' ifadesinden sonra gelen alan adı ile şart içerisinde kullanılacak değerin arasında yer almaktadır. 'Where' ifadesinde kullanılan karşılaştırma operatörlerinin söz dizimi aşağıda verilmiştir.

```
WHERE <AlanAdı> [Karşılaştırma operatörü] <AranacakDeğer>
```

Mantıksal Operatörler

SQL sorgularında kullanılan ve birden fazla koşullar arasındaki ilişkileri ifade etmek için kullanılan operatörlerdir. Bu operatörler, sorgularda belirli koşulların yerine getirilip getirilmediğini değerlendirmek için kullanılmaktadır. Mantıksal operatörler genellikle *WHERE* ifadesi içinde kullanılır.

Mantıksal Operatörler	
AND	Birleştirilen koşulların tümüne uyan kayıtları listeler.
OR	Tanımladığımız koşullardan en az bir tanesine uyan kayıtları listeler.
NOT	Kendisinden sonra gelen koşulu sağlamayan kayıtları listeler.

NOT operatörü, yapılan işlemin tersini kontrol etmektedir. Mantıksal ifade sonucu **True** ise **False** yapar, **False** ise **True** yapar. *LIKE*, *IN* ve *BETWEEN* gibi ifadelerle birlikte kullanılmaktadır.

i Where ifadeleri içerisinde *NOT* anahtar kelimesini içeren koşullar sorguları yavaşlatmaktadır. Zorunlu olmadıkça *NOT* anahtar kelimesi kullanılmamalıdır.

Like

Bir alan içerisinde arama işlemi yapılacaksa 'WHERE' ifadesi ile = (eşittir) sembolü yerine *LIKE* operatörü kullanılmaktadır. *LIKE* ile metinsel alanlar (Varchar, Text, NVarChar) içerisinde arama yapılabilmektedir. *Like* ifadesinin söz dizimi aşağıda verilmiştir.

```
WHERE <AlanAdı> LIKE <AranacakDeğer>
```

Kullanımı	Açıklama
WHERE Adi_Soyadi LIKE 'Me%'	Adı Soyadı Me ile başlayan kayıtları gösterir.
WHERE Adi_Soyadi LIKE '%ih'	Adı Soyadı ih ile biten kayıtları gösterir.
WHERE Adi_Soyadi LIKE '%kar%'	Adı Soyadı içinde kar geçen kayıtları gösterir.
WHERE Adi_Soyadi LIKE '_as'	Üç karakterden oluşan ve son iki karakteri as olan kayıtları gösterir.

Order By

SQL sorgularında belirli bir sıralama düzeni ile sonuçları getirmek için kullanılan bir ifadedir. Bu ifade genellikle *SELECT* sorgularının sonunda almaktadır.

Bir veya birden fazla alana göre sıralama yapılmaktadır. Sıralamanın nasıl yapılacağını belirlemek için *ASC* ve *DESC* kullanılmaktadır. *ASC* kullanılırsa artan yani A'dan Z'ye veya küçükten büyüğe doğru sıralama yapılır. *DESC* kullanılırsa azalan yani Z'den A'ya veya büyükten küçüğe doğru sıralama yapılır. SQL cümlesinde *ORDEY BY* ifadesinden sonra *ASC* veya *DESC* kullanılmaz ise *ASC* olduğu varsayılarak kayıtlar listelenmektedir.

Order By deyimi SQL cümlesinin en sonunda kullanılmaktadır. *WHERE* ifadesi kullanılmış ise *ORDER BY* deyimi *WHERE* ifadesinden de sonra kullanılmaktadır.

Order By ifadesinin söz dizimi aşağıda verilmiştir.

```
SELECT <AlanAdı> FROM <TabloAdı> ORDER BY <AlanAdı1>,<AlanAdı2> ASC veya DESC
```

Belirli Sayıda Veriyi Seçmek (Limit – Offset)

LIMIT ve *OFFSET* ifadeleri, SQL sorgularında sonuç kümesinin sınırlanması ve belirli bir konumdan itibaren sonuçların alınması için kullanılan ifadelerdir.

LIMIT

Bir sorgu sonucu dönen veri kümesinin kaç kayıt içereceğini belirlemektedir. Örneğin, LIMIT 10 ifadesi, sorgu sonucu dönen veri kümesinin ilk 10 kaydını getirecektir.

```
SELECT * FROM müsteriler
LIMIT 10;
```

OFFSET

Sorgu sonucu dönen veri kümesinde hangi konumdan itibaren sonuçları alacağını belirlemektedir. Örneğin, OFFSET 10 ifadesi, sorgu sonucu dönen veri kümesindeki 11. kayıttan başlayarak sonraki kayıtları getirecektir. *OFFSET* genellikle *LIMIT* ile birlikte kullanılmaktadır.

```
SELECT * FROM müsteriler
LIMIT 10 OFFSET 10;
```

Distinct

SQL sorgularında tekrarlanan değerleri ortadan kaldırmak için kullanılan bir ifadedir. Bu ifade, bir sorgu sonucunda dönen veri kümesindeki benzersiz (tekrar etmeyen) değerleri getirmek için kullanılmaktadır.

Örnek kullanım:

```
SELECT DISTINCT sütun1, sütun2, ...
FROM tablo;
```

Bu sorgu, belirtilen sütunlardaki tekrarlayan değerleri bir kez getirir ve sonuç kümesinde her bir kombinasyonun yalnızca bir örneğini içermektedir.

Örneğin, aşağıdaki sorgu müşteri tablosundaki benzersiz şehir isimlerini getirecektir:

```
SELECT DISTINCT sehir
FROM müsteriler;
```

Alanlarda Takma Ad (Alias) Kullanmak

SQL sorgularında belirli bir alanın adını geçici olarak değiştirmek veya kısaltmak için kullanılan bir tekniktir. *AS* kelimesi ile sorgu sonuçlarındaki sütun adlarını daha anlamlı veya okunabilir hale getirmek için kullanılmaktadır.

Örnek kullanım:

```
SELECT sütun1 AS takma_ad1, sütun2 AS takma_ad2, ...
FROM tablo;
```

Bu sorgu, sütun1'in adını "takma_ad1" olarak değiştirir ve sütun2'nin adını "takma_ad2" olarak değiştirir. Sorgu sonuçlarında bu takma adlar kullanılarak veriye erişilebilir.

Örneğin:

```
SELECT müsteri_adı AS ad, müsteri_soyadı AS soyad, müsteri_tel AS telefon
FROM müsteriler;
```

Bu sorgu, "müsteri_adı" sütununu "ad", "müsteri_soyadı" sütununu "soyad" ve "müsteri_tel" sütununu "telefon" olarak tanımlar. Bu şekilde sorgu sonuçları daha anlamlı hale gelir.

Takma adlar aynı zamanda sütunlara matematiksel işlemler uygulandığında veya sütunlardan türetilen ifadeler oluşturulduğunda da kullanılmaktadır.

15.3.2. Insert (Yeni Kayıt Eklemek)

SQL'de *INSERT* ifadesi, bir veri tabanına yeni veri eklemek için kullanılan bir SQL komutudur. *INSERT* komutu, belirtilen bir tabloya yeni bir kayıt (satır) eklemek için kullanılır ve bu kayıtta bulunan değerler, belirtilen sütunlara atanmaktadır.

INSERT ifadesinin genel yapısı şu şekildedir:

```
INSERT INTO <TabloAdı> (AlanAdi1, AlanAdi2, AlanAdi3, ...)
VALUES (Değer1, Değer2, Değer3, ...);
```

Örnek:

```
INSERT INTO müsteriler (müsteri_adı, müsteri_soyadı, müsteri_tel)
VALUES ('Hasan Baki', 'Soy', '555-1234');
```

Bu örnekte, "müsteriler" tablosuna yeni bir müşteri eklenerek bu müşterinin adı, soyadı ve telefon numarası belirtilmiştir.

15.3.3. Delete (Kayıt Silmek)

SQL'deki *DELETE* ifadesi, bir veri tabanındaki kayıtları silmek için kullanılan bir SQL komutudur. *DELETE* komutu, belirtilen bir tablodan belirli bir koşulu sağlayan kayıtları silmek için kullanılmaktadır.

DELETE ifadesinin genel yapısı şu şekildedir:

```
DELETE FROM <TabloAdı>
WHERE koşul;
```

Örnek:

```
DELETE FROM müsteriler
WHERE müsteri_id = 5;
```

Bu örnekte, "müsteriler" tablosundan "müsteri_id" değeri 5 olan müşteriyi silecek bir *DELETE* ifadesi kullanılmıştır.

DELETE ifadesi, genellikle veri tabanında gereksiz, hatalı veya güncelliğini yitirmiş verileri temizlemek için kullanılır. Dikkatlice kullanılmalıdır çünkü yanlış kullanımı veri kaybına neden olabilir. Bu nedenle, silme işlemleri önce dikkatlice düşünülmelidir ve gerektiğinde yedekleme yapılmalıdır.

15.3.4. Update (Kayıt Güncellemek)

SQL'deki *UPDATE* ifadesi, bir veri tabanındaki kayıtların değerlerini güncellemek için kullanılan bir SQL komutudur. *UPDATE* komutu, belirtilen bir tablodaki kayıtların değerlerini belirli bir koşulu sağlayan yeni değerlerle güncellemek için kullanılmaktadır.

UPDATE ifadesinin genel yapısı şu şekildedir:

```
UPDATE <TabloAdı>
SET sütun1 = değer1, sütun2 = değer2, ...
WHERE koşul;
```

Hangi kayıtların güncelleneceğini belirten koşul olan *WHERE* komutu kullanılmalıdır. Bu kısım belirtilmezse, tüm kayıtlar güncellenir.

Örnek:

```
UPDATE müsteriler
SET müsteri_adı = 'YeniAd', müsteri_soyadı = 'YeniSoyad'
WHERE müsteri_id = 5;
```

Bu örnekte, "müsteriler" tablosundaki “müşteri_id” bilgisi 5 olan müşterinin adı ve soyadı güncellenir.

UPDATE ifadesi, veri tabanındaki verileri güncellemek ve kayıtları güncel verilere uyumlu hale getirmek için kullanılmaktadır.

15.4. Veri Tabanı Bağlantıları Temel Özellikleri

Veri tabanı bağlantıları, Clomosy'de uygulama geliştirirken önemli bir rol oynamaktadır. Bu bağlantılar, uygulamalarınızın veri tabanlarına erişimini ve veri manipülasyonunu kolaylaştırmaktadır. İşte Clomosy'de veri tabanı bağlantılarının temel özellikleri:

Open

Veri tabanı sorgusu sonucu elde edilen veri setini (result set) açmak veya kullanıma hazır hale getirmek için *Open* metodu kullanılmaktadır. Bir sorgu çalıştırılmadan önce genellikle bağlantı açılır.

```
Clomosy.DBSQLServerQuery.Open;
```

First

Veri tabanlarından alınan sorgu sonuçları üzerinde hareket etmek için kullanılan *First* fonksiyonu, sorgu sonuç kümesinin ilk kaydına gitmek amacıyla kullanılmaktadır. *First* fonksiyonunun temel amacı, veri tabanındaki kayıtlar arasında gezinmeyi sağlamaktır.

İşlevi ve amacını anlamak için basit bir örnek kullanalım. Örneğin, aşağıdaki gibi bir SQL sorgusu ile bir müşteri listesi çektiğinizi düşünelim:

```
MyQuery.SQL.Text = 'SELECT * FROM Urunler';
MyQuery.Open;
```

Bu sorgu ile tüm müşteri kayıtlarını içeren bir sonuç kümesi elde etmiş olursunuz. Ardından, bu kayıtlar üzerinde gezinmek için *First* fonksiyonunu kullanabilirsiniz:

```
MyQuery.First;
while (not MyQuery.Eof)
{
  // Burada yapılacak işlemler
  // Örneğin, ShowMessage(MyQuery.FieldByName('UrunAdi').AsString);
  MyQuery.Next;
}
```

Bu örnekte, *First* fonksiyonu ile sorgu sonuç kümesinin ilk kaydına gidilir ve ardından while döngüsü içinde *Eof* (End of File) fonksiyonu ile sona gelinene kadar kayıtlar üzerinde dönülür. Bu şekilde, veri tabanındaki kayıtlar arasında gezinme ve her bir kayıt üzerinde işlem yapma imkanına sahip olunur.

Next

Next fonksiyonu, veri tabanı sorgusu sonuç kümesinde bir sonraki kayda geçmek için kullanılmaktadır. Bu fonksiyon, sorgu sonuç kümesinde bir sonraki kayda geçtiği zaman **True** değerini döndürür. Eğer sorgu sonuç kümesinde bir sonraki kayıt yoksa (yani sona gelinmişse), *Next* fonksiyonu **False** (Doğru) değerini döndürmektedir.

Örnek bir kullanım:

```
MyQuery.First; // İlk kayda gidilir

while (not MyQuery.Eof)
{
  // Burada yapılacak işlemler
  // Örneğin, ShowMessage(MyQuery.FieldByName('UrunAdi').AsString);
  MyQuery.Next; // Bir sonraki kayda geç
}
```

While döngüsü içinde MyQuery.Eof fonksiyonu ile sona gelinene kadar her bir kayıt üzerinde işlem yapılarak 'MyQuery.Next' fonksiyonu ile bir sonraki kayda geçilmektedir.

Bu şekilde, veri tabanı sorgusu sonuç kümesi üzerinde gezinme ve her bir kayıt üzerinde işlem yapma işlemleri gerçekleştirilebilir.

Last

Last fonksiyonu, bir *dataset* türetilmiş bileşen üzerindeki kayıtlar arasında gezinme işlemlerini kontrol etmek için kullanılmaktadır. *Last* fonksiyonu, sorgu sonuç kümesindeki kayıtlar arasında en sona gidilmesini sağlamaktadır. Bu fonksiyonun kullanılması, sorgu sonuç kümesinin en sonundaki kayıtları kontrol etmek için yaygındır.

Aşağıda basit bir örnek kullanım bulunmaktadır:

Örnekte veri tabanında 'UrunBilgisi' tablosunda bulunan veriler sırasıyla şöyledir:

Kalem – Defter – Silgi – Kutu

Aşağıdaki kodda *urunAdi* alanının mesaj kutusuyla alınmasından önce *Last* fonksiyonu yazılırsa mesaj kutusunda sadece **Kutu** değeri döndürülür. Mesaj kutusundan sonra yazılır ise sadece **Defter** değeri döndürülür.

```
var
  VeriSorgusu : TClSQLQuery;
    {
  Clomosy.DBSQLServerConnect('SQL Server', 'sunucu_adi', 'kullanici_adi',
'kullanici_sifre','veritabani_adi', 1433);

  try
    VeriSorgusu = Clomosy.DBSQLServerQueryWith('SELECT * FROM UrunBilgisi');
    VeriSorgusu.Open;
    while (not VeriSorgusu.Eof)
    {
      VeriSorgusu.Last;
      ShowMessage(VeriSorgusu.FieldByName('urun_adi').AsString);
      //VeriSorgusu.Last;
    }
  finally
    VeriSorgusu.Free;
  }
}
```

Prior

Prior fonksiyonu, bir veri seti türetilmiş bileşen üzerindeki kayıtlar arasında geriye doğru (önceki kayıtlara) gezinme işlemlerini kontrol etmek için kullanılmaktadır. *Prior* fonksiyonu, sorgu sonuç kümesindeki kayıtlar arasında bir önceki kayıta gidilmesini sağlar.

Free

Bellekte ayrılmış veri tabanı nesnesini serbest bırakmak için kullanılır. Bellek yönetimi açısından önemlidir.

```
MyQuery.Free;
```

Close

Veri tabanı bağlantısını kapatmak için kullanılmaktadır. Bağlantıyı kapatmadan önce kullanımı bitmiş nesnelerin bellekten serbest bırakılması güvenlik açısından önemlidir.

```
MyQuery.Close;
```

Found

Veri seti nesnesinin, içerisindeki kayıtlardan en az bir tanesinin bulunup bulunmadığını kontrol etmek için **Found** metodu kullanılmaktadır. Bu özellik, bir sorgunun sonucunda elde edilen veri kümesinde kayıt varsa True, yoksa False (Doğru) değerini döndürmektedir.

```
If (not Clomosy.DBSQLServerQuery.Found)
```

EOF

Veri tabanı sorgusu sonuç kümesindeki kayıtların sona erip ermediğini kontrol etmek için kullanılmaktadır. *EOF* durumu, bir sorgu sonuç kümesinde bir sonraki kaydolup olmadığını belirtmektedir. Eğer sorgu sonuç kümesinde bir sonraki kayıt yoksa, *EOF* durumu True değerini alır; aksi takdirde, False (Doğru)değerini alır.

```
Qry = Clomosy.DBSQLiteQueryWith('SELECT * FROM tablo');
  Qry.OpenOrExecute;
  while (not Qry.Eof ) {
    ShowMessage(Qry.FieldByName('veri1').AsString);
    Qry.Next;
  }
```

RecordCount

Veri setinin içerisinde kaydolup olmadığına bakmak için başka bir yöntemde kullanılır. Bu yöntem **RecordCount** metodudur. Bu metot kayıt sayısını döndürmektedir.

```
If (Clomosy.DBSQLServerQuery.RecordCount > 0)
```

Edit

Veri tabanı sorgusu sonucunda elde edilen veri kümesi (result set) üzerindeki kayıtların düzenlenmesi için **Edit** metodu kullanılmaktadır. *Edit* metodu, bir kaydın değerlerini değiştirmek ve bu değişiklikleri kaydedebilmek için önemlidir.

```
Clomosy.DBSQLServerQuery.Edit;
```

Insert

Veri tabanı içerisindeki kayıt (record) eklemek için **Insert** metodu kullanılmaktadır. Bu metot, genellikle bir veri tabanı tablosuna yeni bir kayıt eklemeyi sağlamaktadır.

```
Clomosy.DBSQLServerQuery.Insert;
```

Post

Tablo üzerinde ekleme ve düzenleme için veri seti üzerinde alan adlarına göre atamalar yapılmaktadır. Ardından tablodaki düzenleme veya ekleme modundan çıkarak değişiklikleri kalıcı hale getirmek amacıyla **Post** metodu kullanılmaktadır.

```
Clomosy.DBSQLServerQuery.FieldByName('model').AsString = 'Alpha';
Clomosy.DBSQLServerQuery.FieldByName('urun_fiyati').AsInteger = 104;
Clomosy.DBSQLServerQuery.Post;
```

Bu şekilde bir tabloda değişiklikler yapılabilmektedir.

SQL

SQL sorgusunu belirlemek veya almak için kullanılmaktadır. Bu özellik, sorguların dinamik olarak oluşturulması veya kontrol edilmesi için önemlidir.

```
MyQuery.SQL.Text = 'SELECT * FROM Urunler';
```

FieldByName

SQL sorgusunda kullanılan parametre değerlerine erişim sağlamak için kullanılmaktadır. Parametreler, dinamik sorgular ve güvenli sorgular oluşturmaktadır.

```
Clomosy.DBSQLServerQuery.FieldByName('id').AsString = '13';
```

```
{
  Clomosy.DBSQLServerConnect('SQL Server', 'sunucu_adi', 'kullanici_adi',
'kullanici_sifre','veritabani_adi', 1433);

  try
    Clomosy.DBSQLServerQuery.Sql.Text = 'SELECT * FROM UrunBilgisi WHERE
urun_fiyati=103';
    Clomosy.DBSQLServerQuery.Open;

    If (Clomosy.DBSQLServerQuery.RecordCount > 0)
    {
      Clomosy.DBSQLServerQuery.Edit;
      Clomosy.DBSQLServerQuery.FieldByName('model').AsString = 'Alpha';
      Clomosy.DBSQLServerQuery.FieldByName('urun_fiyati').AsInteger = 104;
      Clomosy.DBSQLServerQuery.Post;
     }
  except
    ShowMessage('Exception Class: '+LastExceptionClassName+' Exception Message:
'+LastExceptionMessage);
  }
}
```

ExecSQL

Sorgu yapısını alarak ekleme, güncelleme, silme vb. SQL ifadelerini alarak işlem yapılmak isteniyorsa **ExecSQL** metodu da tercih edilmektedir. Bu metot bir SQL sorgusunu yürütmek için kullanılmaktadır. Bu sorgu, genellikle veri tabanında değişiklik yapacak (insert, update, delete gibi) bir SQL ifadesidir. Yani sorgu sonucunda bir veri seti dönmeyi beklemez, ancak veri tabanında bir değişiklik yapar.

```
Clomosy.DBSQLServerQuery.Sql.Text = 'INSERT INTO UrunBilgisi (model, urun_fiyati)
                                    VALUES ('+QuotedStr(' Raspberry')+',9999)';
Clomosy.DBSQLServerQuery.ExecSQL;
```

15.5. Local (Yerel) Veri Tabanı

Local veri tabanı, genellikle bir bilgisayarın, bir uygulamanın yerel belleğinde veya dosya sisteminde barındırılan küçük çaplı bir veri tabanıdır. Local veri tabanları genellikle tek bir kullanıcının ihtiyaçlarını karşılamak üzere tasarlanmıştır ve genellikle uygulamanın veya sistem bileşeninin içinde bulunmaktadır.

Local veri tabanlar, genellikle hafif ve düşük kaynak tüketimli olup, küçük ölçekli uygulamalar için idealdir. Local veri tabanları, genellikle hızlı erişim ve yüksek performans sağlamaktadır. Çünkü veri tabanı sorguları yerel makinede çalışır ve ağ gecikmeleri olmaz.

Local veri tabanlar, SQLite, Microsoft Access, Firebird Embedded ve benzeri teknolojilerle sağlanmaktadır. Bu veri tabanlar, genellikle hafif, taşınabilir ve kullanımı kolaydır.

SQLite

SQLite, hafif, taşınabilir ve tek kullanıcılı bir SQL tabanlı veri tabanı motorudur. Bu, bir sunucu gerektirmeyen, yerel olarak çalışabilen bir veri tabanı çözümüdür. SQLite, genellikle gömülü sistemler, masaüstü uygulamaları ve mobil uygulamalar gibi hafif ve taşınabilir uygulamalarda kullanılmak üzere tasarlanmıştır.

SQLite, küçük bir hafıza ayak izine sahip olan ve herhangi bir sunucu gerektirmeyen bir veri tabanı çözümüdür. Bu özellikleri sayesinde uygulamalar içinde kolayca entegre edilmektedir.

SQLite, aynı anda sadece bir kullanıcının yazma işlemini gerçekleştirebildiği bir tek kullanıcılı bir veri tabanıdır. Bu nedenle, çok kullanıcılı büyük sistemler için uygun olmayabilir.

15.5.1. Veri Taban Oluşturmak

Araç çubuğunda yer alan **Yeni Veritabanı** butonuna tıklayın. Oluşturulacak yeni veri tabanı yolu için bir pencere açılacaktır.

Bu pencerede veri tabanının oluşturulacağı dizini seçin ve **Dosya Adı** alanına veri tabanı adını yazıp **Aç** butonuna tıklayın.

SQLite Expert yönetim aracının veri tabanı listeleme penceresine, yeni oluşturulan veri tabanı eklenecek ve ana penceredeki **Veritabanı Yapısı** alanında oluşturulan veri tabanı bilgileri görüntülenecektir.

15.5.2. Tablo ve Tablo Alanları Oluşturmak

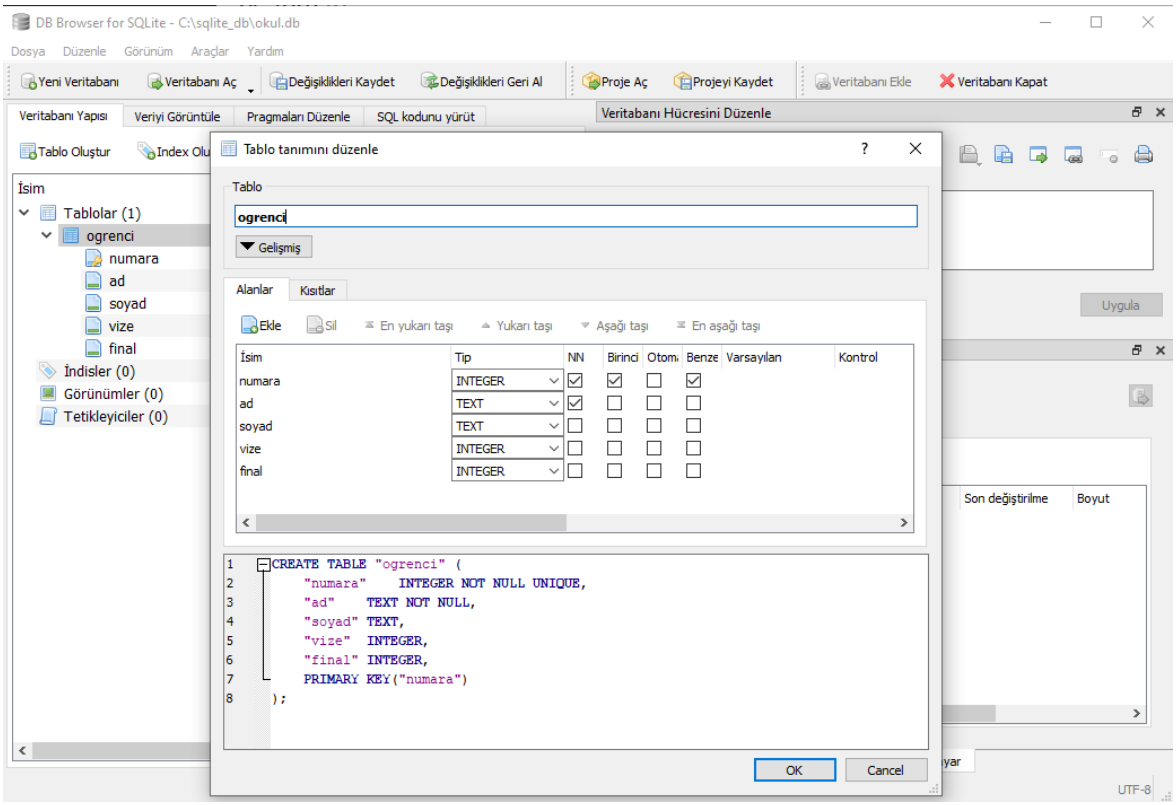
Veri tabanı üzerinde bulunan tablo yapıları, asıl verilerin kaydedildiği kısımdır. Bundan dolayı tabloların ilk oluşum aşamasında tablo yapısı ve alanlarının iyi analiz edilmesi ve bu doğrultuda modellendirilmesi son derece önemlidir.

Bir örnekle veri tabanı yönetim aracı ve tablo yapısı daha iyi anlaşılacaktır.

Tablo oluşturmak için veri tabanı penceresinde daha önceden oluşturulan **Okul** veri tabanı seçilir ve araç çubuğunda yer alan **Tablo Oluştur** butonuna tıklanır.

Ana ekranda açılan yeni pencerede **Tablo** alanına tablo adı yazılır. Örnekte **ogrenci** tablo adı kullanılmıştır.

Tablo üzerinde alanlar oluşturmak için **Alanlar** sekmesine geçilir. Yeni bir alan eklemek için sol kısmında yer alan **Ekle** butonuna tıklanır. Açılan pencerede bir alana ait birçok özellik belirlenmektedir. Tablo alanında sütunlar bulunmaktadır. Burada **İsim** kısmında bir isim verilir ve **Tip** alanına ise isim alanı değişkeninin veri tipi girilir. Örnek bir ad girilecekse **TEXT** olarak seçilmeli veya numara girilecekse **INTEGER** seçilmelidir. İhtiyaç halinde diğer özellikleri seçerek **OK** butonuna tıklanır.



Resim 68

Yukarıdaki örnekte **okul** veri tabanına "**ogrenci**" tablosu oluşturulmuştur. Burada beş alan tanımlanmış ve veri tipleri belirtilerek istenilen özellikler seçilmiştir.

SQLite veri tabanında, veriler üzerinde işlemler yapmadan önce uygulamada ***TClSQLiteQuery*** sınıfına ait bir nesne oluşturulmalıdır.

Kayıt işlemleri *DBSQLServerQueryWith* veya *DBSQLiteQuery* fonksiyonları kullanılmaktadır.

SQLite bağlantılarının nasıl kullanıldığına dair bilgiler *Clomosy Kütüphanesi* konu başlığı altındaki *Veri Tabanı Erişimi* alt başlığı altında *Local (Yerel) Veri Taban Bağlantıları* anlatılmıştır.

SQL yapılarının kullanımını yukarıda oluşturulan "**ogrenci**" tablosu üzerinden kullanacağız.

Tablo Görüntüleme (SELECT)

Oluşturulan veri tabanındaki "**ogrenci**" tablosunu görüntülemek için *TClSQLiteQuery* sınıfına ait bir nesne üzerinden görüntüleme yapılmaktadır. *DBSQLiteQueryWith* ile ya da *DBSQLiteQuery* sorgusu ile veriler çekilmektedir.

Aşağıdaki örnekte "**ogrenci**" tablosundaki vize ve final ortalamaları üzerinden belirli bir kriteri karşılayan öğrencileri listelemek için SQL sorgusu oluşturulur. Bu sorguda vize %40 ve final %60 oranında değerlendirilmektedir ve 87 üzeri not alan öğrenciler listelenmektedir:

```
 var
  Sifre,DB  : String;
{
  Sifre = '';
  DB = 'C:\sqlite_db\okul.db';
  Clomosy.DBSQLiteConnect(DB, Sifre);

  try
    Clomosy.DBSQLiteQuery.Sql.Text = 'SELECT * FROM ogrenci WHERE (vize * 0.4 +
final * 0.6) > 87';
    Clomosy.DBSQLiteQuery.OpenOrExecute;
    while (not Clomosy.DBSQLiteQuery.Eof)
    {
      ShowMessage(Clomosy.DBSQLiteQuery.FieldByName('ad').AsString);
      Clomosy.DBSQLiteQuery.Next;
    }

  except

    ShowMessage('Exception Class: '+LastExceptionClassName+' Exception Message:
'+LastExceptionMessage);
  }
}
```

Tabloya Veri Eklemek (INSERT)

INSERT ifadesi ile tabloya yeni kayıt eklenmektedir. Insert direktifi, **Into** ve **Values** SQL sözcükleri ile kullanılmaktadır.

Aşağıdaki örnekte vize ve final notları üzerinde hesaplanan ortalamada 87 üstünde not olan bir öğrenci var ise düzenleme (*Edit*) yok ise ekleme (*Insert*) işlemi yapılmaktadır.

```
var
  Sifre,DB  : String;
{
  Sifre = '';

  DB = 'C:\sqlite_db\okul.db';

  Clomosy.DBSQLiteConnect(DB, Sifre);

  try
    Clomosy.DBSQLiteQuery.Sql.Text = 'SELECT * FROM ogrenci WHERE (vize * 0.4 +
final * 0.6) > 87';
    Clomosy.DBSQLiteQuery.OpenOrExecute;
    if (Clomosy.DBSQLiteQuery.Found)
    {
      //Clomosy.DBSQLiteQuery.Sql.Text = 'INSERT INTO ogrenci (numara, ad, soyad, vize,
final) VALUES (127, "Ahmet", "Yılmaz", 75, 85)';
      //Clomosy.DBSQLiteQuery.Exec;
     Clomosy.DBSQLiteQuery.Edit;
    }else
      Clomosy.DBSQLiteQuery.Insert;

    Clomosy.DBSQLiteQuery.FieldByName('vize').AsInteger = 90;
    Clomosy.DBSQLiteQuery.FieldByName('final').AsInteger = 90;
    Clomosy.DBSQLiteQuery.Post;

  except

    ShowMessage('Exception Class: '+LastExceptionClassName+' Exception Message:
'+LastExceptionMessage);
  }
}
```

Tablodaki Verileri Güncelleme (UPDATE)

Tablolar üzerindeki kayıtlarda güncelleme işlemi *Update* ifadesi ile gerçekleştirilir. Update işleminde **SET** ifadesi ile güncellenecek alanlar ve bu alanların yeni değerler belirtilmektedir.

Örnekte; vize puanları 70 üstü olan öğrenci var ise final puanlarına 15 puan eklemek için aşağıdaki gibi bir *UPDATE* SQL sorgusu kullanılabilir:

```
var
  Sifre,DB  : String;
{
  Sifre = '';
  DB = 'C:\sqlite_db\okul.db';
  Clomosy.DBSQLiteConnect(DB, Sifre);

  try
    Clomosy.DBSQLiteQuery.Sql.Text = 'SELECT * FROM ogrenci WHERE vize >= 70';
    Clomosy.DBSQLiteQuery.OpenOrExecute;
    if (Clomosy.DBSQLiteQuery.Found)
    {
      //Clomosy.DBSQLiteQuery.Sql.Text = 'UPDATE ogrenci SET final = final + 5';
      //Clomosy.DBSQLiteQuery.Exec;
      Clomosy.DBSQLiteQuery.Edit;
      Clomosy.DBSQLiteQuery.FieldByName('final').AsInteger =
Clomosy.DBSQLiteQuery.FieldByName('final').AsInteger + 15;
      Clomosy.DBSQLiteQuery.Post;
    }

  except

    ShowMessage('Exception Class: '+LastExceptionClassName+' Exception Message:
'+LastExceptionMessage);
  }
}
```

Tablodan Veri Silme (DELETE)

Tablodan bir veya daha fazla kayıt silmek için *Delete* ifadesi, bütün kayıtları silmemek için *Delete* ifadesi ile **Where** kullanılmaktadır. **Where** ifadesi kullanıldığında belirtilen şartları sağlayan kayıtlar silinir.

"**ogrenci**" tablosunda, vize ve final puanları üzerinden alınan ortalama bilgisi 60 altında olan öğrencileri silen bir SQL sorgusu aşağıdaki gibidir.

```
var
  Sifre,DB  : String;
{
  Sifre = '';

  DB = 'C:\sqlite_db\okul.db';

  Clomosy.DBSQLiteConnect(DB, Sifre);

  try
    Clomosy.DBSQLiteQuery.Sql.Text = 'DELETE FROM ogrenci WHERE (vize * 0.4 +
final * 0.6) < 60';
    Clomosy.DBSQLiteQuery.Exec;
```

```
  except

   ShowMessage('Exception Class: '+LastExceptionClassName+' Exception Message: '+LastExceptionMessage);
  }
}
```

15.6. SQL Server

Microsoft SQL Server, Microsoft tarafından geliştirilen ve genellikle büyük ölçekli kurumsal veri tabanları yönetmek için kullanılan bir ilişkisel veri tabanı yönetim sistemidir (RDBMS). SQL Server, Windows işletim sistemlerinde çalışarak işletmelerin veri depolama, işleme ve yönetme ihtiyaçlarına çeşitli hizmetler sunmaktadır.

SQL Server, büyük miktarlardaki veriyi güvenli bir şekilde depolamanıza olanak tanımaktadır. Tablolar, sütunlar ve satırlar aracılığıyla veri tabanında yapılandırılmış verileri barındırmaktadır. SQL Server, T-SQL (Transact-SQL) adı verilen özel bir SQL dilini kullanarak veriler üzerinde sorgular çalıştırmanıza, saklamanıza ve işlemenize olanak tanır. İşlemler, veri eklemek, güncellemek, sorgulamak ve silmek gibi çeşitli veri manipülasyon işlemlerini içerir.

SQL Server, geniş bir kullanıcı kitlesine hitap eden bir veri tabanı çözümüdür ve büyük ölçekli kurumsal uygulamalardan küçük işletmelere kadar birçok alanda kullanılmaktadır.

Clomosy, kullanıcılarına güçlü bir SQL Server entegrasyonu sunarak veri tabanı işlemlerini daha etkili ve kullanıcı dostu hale getiren bir platformdur. SQL Server ile entegre çalışma yeteneği, Clomosy'i iş süreçlerinizi daha verimli bir şekilde yönetmenizi sağlayan kapsamlı bir çözüm haline getirmektedir.

15.6.1. Veri Tabanı Oluşturmak

SQL Server üzerinde veri tabanı oluşturmak için birkaç farklı yöntem bulunmaktadır. İşte bu yöntemlerden bazıları:

SQL Server Management Studio (SSMS) Kullanarak:

SQL Server Management Studio'yu açın. Sol taraftaki **Object Explorer** penceresinde **Databases** klasörünü genişletin. Sağ tıkladıktan sonra **New Database** seçeneğini seçin. Açılan pencerede veri tabanının adını ve gerekirse diğer ayarları belirtin. **OK** düğmesine tıklayarak veri tabanını oluşturun.

T-SQL Komutları ile:

Yeni bir sorgu penceresi açın. Aşağıdaki gibi bir T-SQL sorgusu kullanarak veri tabanını oluşturun:

```
CREATE DATABASE [VeritabaniAdi]
```

VeritabaniAdi kısmını oluşturmak istediğiniz veri tabanının adıyla değiştirin.

Bu yöntemlerden birini kullanarak SQL Server üzerinde yeni bir veri tabanı oluşturulmaktadır. Özellikle SQL Server Management Studio'nun grafik arayüzü, kullanıcı dostu bir şekilde veri tabanı oluşturmayı sağlar.

15.6.2. Tablo ve Tablo Alanları Oluşturmak

SQL Server üzerinde yeni bir tablo oluşturmak ve tabloya alan eklemek için aşağıdaki adımları izlenebilir:

SQL Server Management Studio (SSMS) Kullanarak:

Sol taraftaki **Object Explorer** penceresinde bulunan veri tabanınızı genişletin. **Tables** klasörüne sağ tıklayın ve **New - Table** seçeneğine tıklayarak yeni bir tablo tasarımı açın.

Açılan pencerede tablonuzun alanlarını ekleyin. Alan adını, veri tipini, uzunluğunu ve diğer özellikleri belirtin. Tasarım penceresinde sol üst kısımda **Save** butonuna tıklayarak veya Ctrl+S yaparak tabloyu kaydedin. Verilen isim ile tablo oluşturulur.

Tablo oluşturma örneği için bir "Ürünler" tablosu oluşturalım. Bu tablo, bir e-ticaret sitesinde bulunan ürünlerin bilgilerini içerecektir.

Column Name	Data Type	Allow Nulls
UrunID	int	☐
UrunAdi	varchar(100)	☐
Kategori	varchar(50)	☑
Fiyat	decimal(10, 2)	☑
StokMiktari	int	☑
SonGuncellemeTarihi	date	☑
		☐

Resim 69

Bu işlem, "Urunler" adında bir tablo oluşturur ve bu tablonun alanlarını belirtir. Yukarıda oluşturulan tablonun alanları ve veri tipleri:

UrunID: Ürünlerin benzersiz kimlik numarası olarak kullanılacak tamsayı (INT) alanı ve aynı zamanda birincil anahtar (PRIMARY KEY).

UrunAdi: Ürünün adını içeren bir metin (VARCHAR) alanı. NOT NULL, bu alanın boş olamayacağını belirtir.

Kategori: Ürünün ait olduğu kategoriyi içeren bir metin alanı.

Fiyat: Ürünün fiyatını tutan bir ondalık sayı (DECIMAL) alanı.

StokMiktari: Ürünün stok miktarını tutan bir tamsayı (INT) alanı.

SonGuncellemeTarihi: Ürün bilgilerinin son güncellenme tarihini tutan bir tarih (DATE) alanı.

T-SQL Komutları ile:

Yukarıdaki tabloyu SQL komutu ile oluşturalım.

Veri tabanına bağlanın. Sol üst köşede bulunan **New Query** butonuna tıklayarak yeni bir sorgu penceresi açalım. Aşağıdaki gibi bir SQL sorgusu ile yeni bir tablo oluşturulur:

```
CREATE TABLE Urunler
(
  UrunID INT PRIMARY KEY,
  UrunAdi VARCHAR(100) NOT NULL,
  Kategori VARCHAR(50),
  Fiyat DECIMAL(10, 2),
  StokMiktari INT,
  SonGuncellemeTarihi DATE
)
```

Clomosy platformu üzerinde SQL Server üzerinde oluşturulan tablo ile ilgili select, ınsert, update, delete işlemlerini gerçekleştirelim.

Sql Server bağlantılarının nasıl kullanıldığına dair bilgiler *Clomosy Kütüphanesi* konu başlığı altındaki *Veri Tabanı Erişimi* alt başlığı altında anlatılmıştır.

Tablo Görüntüleme (SELECT)

Oluşturulan veri tabanındaki *Urunler* tablosunu görüntülemek için TClSqlQuery sınıfına ait bir nesne üzerinden görüntüleme yapılmaktadır. *DBSQLServerQueryWith* ile ya da *SQL* sorgusu ile veriler çekilebilir.

Aşağıdaki örnekte *Urunler* tablosundaki *StokMiktari* 40 üstü olan SQL sorgusu oluşturarak verileri getirdi.

```
Var
  urunBilgisiQry : TClSqlQuery;
{
  urunBilgisiQry = TClSqlQuery.Create(nil);
  Clomosy.DBSQLServerConnect('SQL Server', 'sunucu_adi', 'kullanici_adi',
'kullanici_sifre','veritabani_adi', 1433);

  try
    urunBilgisiQry.Connection = Clomosy.DBSQLServerConnection;
    urunBilgisiQry.SQL.Text = 'SELECT * FROM Urunler WHERE StokMiktari > 40';
    urunBilgisiQry.ExecSql;
```

```
  if (urunBilgisiQry.Found)
  {
    ShowMessage(Clomosy.DBDataSetGetAsJSON(urunBilgisiQry));
  }
 finally

  urunBilgisiQry.Free;
 }
}
```

DBSQLServerQueryWith fonksiyonu ile sorgu çekilerek işlemler yapılmaktadır.

```
var
VeriSorgusu : TClSQLQuery;
{
 VeriSorgusu = TClSqlQuery.Create(nil);
 Clomosy.DBSQLServerConnect('SQL Server', 'sunucu_adi', 'kullanici_adi',
'kullanici_sifre','veritabani_adi', 1433);

 try
  VeriSorgusu.Connection = Clomosy.DBSQLServerConnection;
  VeriSorgusu = Clomosy.DBSQLServerQueryWith('SELECT * FROM Urunler WHERE
StokMiktari > 40');
  VeriSorgusu.ExecSql;

  ShowMessage(VeriSorgusu.RecordCount);
  if (VeriSorgusu.RecordCount > 0)
  {
    ShowMessage(Clomosy.DBDataSetGetAsJSON(VeriSorgusu));
  }
 finally
  VeriSorgusu.Free;
 }
}
```

Tabloya Veri Eklemek (INSERT)

Tabloya yeni kayıt eklemek için *INSERT* ifadesi kullanılmaktadır. Insert direktifi, **Into** ve **Values** sql sözcükleri ile kullanılır.

SQL Server veri tabanına bağlanarak ‘Urunler’ tablosuna yeni bir kayıt eklemeyi ve ardından bağlantıyı serbest bırakma işlemi gerçekleştirelim.

```
var
 urunlerQry : TClSqlQuery;
{
 urunlerQry = TClSqlQuery.Create(nil);
```

```
  Clomosy.DBSQLServerConnect('SQL Server', 'sunucu_adi', 'kullanici_adi',
'kullanici_sifre','veritabani_adi', 1433);
  urunlerQry.Connection = Clomosy.DBSQLServerConnection;

  try
    Clomosy.DBSQLServerQuery.Sql.Text = 'INSERT INTO Urunler
(UrunID,UrunAdi,Kategori,Fiyat,StokMiktari,SonGuncellemeTarihi) VALUES
(12,'+QuotedStr('IPad')+','+QuotedStr('Elektronik')+',5000,4,'+QuotedStr(FormatDateTime(
'yyyy-mm-dd', Now))+')';
    Clomosy.DBSQLServerQuery.ExecSQL;

  finally

    urunlerQry.Free;
  }
}
```

Tablodaki Verileri Güncelleme (UPDATE)

Tablolar üzerindeki kayıtlarda güncelleme işlemi *Update* ifadesi ile gerçekleştirilmektedir. Update işleminde **SET** ifadesi ile güncellenecek alanlar ve bu alanların yeni değerleri belirtilmektedir.

Güncelleme sorgusu ile *Urunler* tablosundaki stok miktarı 30 olan kaydın fiyatını 1000 TL arttırma işlemi şu şekilde yapılabilir:

```
var
  urunlerQry : TClSqlQuery;
{
  urunlerQry = TClSqlQuery.Create(nil);
  Clomosy.DBSQLServerConnect('SQL Server', 'sunucu_adi', 'kullanici_adi',
'kullanici_sifre','veritabani_adi', 1433);
  urunlerQry.Connection = Clomosy.DBSQLServerConnection;

  try
    Clomosy.DBSQLServerQuery.Sql.Text = 'UPDATE Urunler SET Fiyat = Fiyat + 1000
WHERE StokMiktari = 30';
    Clomosy.DBSQLServerQuery.ExecSQL;

  finally

    urunlerQry.Free;
  }
}
```

Tablodan Veri Silme (DELETE)

Tablodan bir veya daha fazla kayıt silmek için *Delete* ifadesi, bütün kayıtları silmemek için *Delete* ifadesi ile **Where** kullanılmaktadır. **Where** ifadesi kullanıldığında belirtilen şartları sağlayan kayıtlar silinir.

Urunler tablosundan **Elektronik** kategorisindeki ve stok miktarı **30**'dan az olan verileri silen bir SQL sorgusu aşağıdaki gibidir.

```
var
  urunlerQry : TClSqlQuery;
{
  urunlerQry = TClSqlQuery.Create(nil);
  Clomosy.DBSQLServerConnect('SQL Server', 'sunucu_adi', 'kullanici_adi',
'kullanici_sifre','veritabani_adi', 1433);
  urunlerQry.Connection = Clomosy.DBSQLServerConnection;
  try

    Clomosy.DBSQLServerQuery.Sql.Text = 'DELETE FROM Urunler WHERE Kategori =
"Elektronik" AND StokMiktari <= 30';
    Clomosy.DBSQLServerQuery.ExecSQL;
  finally

    urunlerQry.Free;
  }
}
```

15.7. Cloud (Bulut) Veri Tabanı

Cloud veri tabanları, verilerinizi çevrim içi olarak depolamanın, yönetmenin ve erişmenin güvenilir bir yolunu sunmaktadır. Geleneksel veri tabanı sistemlerine kıyasla, bulut tabanlı çözümler daha fazla esneklik, ölçeklenebilirlik ve erişim kolaylığı sağlamaktadır.

Cloud veri tabanları, işletmelerin ihtiyaçlarına göre esnek bir şekilde ölçeklendirilmektedir. Veri miktarı arttıkça veya azaldıkça, hizmet sağlayıcı bu değişikliklere hızlı bir şekilde yanıt verebilir. Bu, kaynakları optimize etmeyi ve maliyetleri düşürmeyi sağlamaktadır. Bulut veri tabanları genellikle otomatik yedekleme ve kurtarma özelliklerine sahiptir. Bu, veri kaybını önler ve olası bir kesinti durumunda verilerin hızla kurtarılması sağlamaktadır.

Clomosy platformu, kullanıcılara çeşitli iş süreçlerini kolaylaştırmak için hazır tablolar sunmaktadır. Bu hazır tablolara "template" adı verilir. Bu template'ler, belirli iş alanlarına özgü veri yapıları ve ilişkiler içermektedir. Kullanıcılar, bu hazır tablolar üzerinden işlemler yaparak kendi uygulamalarını hızlı bir şekilde oluşturabilir.

i	Clomosy hazır tablolar ve tablo alanlarına dair ayrıntılı bilgiler *Clomosy Şablon (Template) Yapısı* konu başlığı altında anlatılmıştır. Yapılan uygulamaya özgü istenilen tablo kullanılabilir.

Template Kullanımı: Clomosy'nin sağladığı hazır tablolar, çeşitli sektörlere yönelik iş çözümlerini desteklemektedir. Kullanıcılar, bu template'leri kullanarak kendi veri tabanı ihtiyaçlarına uygun uygulamalar geliştirebilir.

Veri İşlemleri: Hazır tablolar üzerinde standart veri işlemleri gerçekleştirilmektedir. Veri ekleme, güncelleme, silme gibi temel işlemler, kullanıcıların ihtiyacına göre uyarlanabilir.

Performans ve Güvenlik: Clomosy platformu, hazır tabloların performansını ve güvenliğini optimize etmek için çeşitli araçlar ve yönergeler sunmaktadır.

Hazır tablolar ve template'ler, Clomosy platformunu kullanarak veri tabanı tabanlı uygulama geliştirmeyi hızlı, güvenilir ve ölçeklenebilir hale getirmektedir.

i	Clomosy hazır tablolar üzerinde bir uygulama yapılacak ise *Cloud (Bulut) Veri Tabanı Bağlantıları* bölümüne bakılarak işlemler gerçekleştirilebilir.

BÖLÜM 16. Metin Dosyası Kullanımı

Clomosy'de metin dosyaları tanımlama, metin tabanlı veri depolamak veya okuma/yazma işlemleri gerçekleştirmek amacıyla kullanılmaktadır. Metin dosyaları, genellikle insanlar tarafından okunabilir ve düzenlenebilir formatlarda veri içermektedir. Clomosy platformunda metin dosyaları ile çalışmak için *TextFile* türünden değişkenler mevcuttur.

Clomosy platformunda iki tip dosya işlemleri bulunmaktadır. Bu standart dosya işlemleri ve özelleştirilmiş dosya işlemleridir.

16.1. Standart Dosya İşlemleri

16.1.1. AssignFile

AssignFile işlevi, Clomosy uygulamasında bir dosya değişkenini bir dosya adıyla ilişkilendirmek için kullanılır. Yani programın belirli bir dosya ile etkileşime girmesini sağlamaktadır. *AssignFile* ile dosya değişkeni oluşturulup belirtilen dosya adıyla ilişkilendirilme yapılmaktadır. Fonksiyon iki parametrelidir. İlk parametreye oluşturulan *TextFile* nesnesine ait değişken, ikinci parametreye ise oluşturulacak dosya adı yazılmalıdır.

Örnek kullanımı:

```
AssignFile(myFile, 'dosya.txt');
```

16.1.2. clFileExists

clFileExists fonksiyonu, belirtilen dosya yolunda bir dosyanın var olup olmadığını kontrol eden bir Clomosy işlevidir. Bu fonksiyon, belirtilen dosyanın mevcut olup olmadığını kontrol eder ve dosya varsa **True**, yoksa **False** değerini döndürmektedir. Fonksiyon bir parametre almaktadır. Bu parametre string veri türünde dosya yolunun yazılmasıdır. Dosya Clomosy

uygulamasının kullanıldığı dosya içerisinde yazılacak ise sadece dosya adı, farklı bir konumda veri sorgulanacak ise dosya yolu ile beraber yazılmalıdır.

Örnek kullanımı:

```
if (FileExists('dosya.txt')) // 'C:\Belgelerim\dosya.txt'
  ShowMessage('Dosya mevcut')
else
  ShowMessage('Dosya mevcut değil');
```

16.1.3. ReWrite

ReWrite, Clomosy prosedürüdür. Dosyayı yeniden yazma (write) modunda açma; dosyanın içeriğini silme ve yeni verileri yazmaya hazır hale getirmektedir.

Prosedür bir parametre alır ve bu *TextFile* nesnesine ait değişken adıdır.

Örnek kullanımı:

```
var
yeniDosya: TextFile;

{
  if (not clFileExists('dosya.txt'))
  {
    AssignFile(yeniDosya, 'dosya.txt');
    Rewrite(yeniDosya);
    WriteLn(yeniDosya, 'yeni');
    WriteLn(yeniDosya, 'World');
  }

    CloseFile(yeniDosya);

}
```

16.1.4. WriteLn

WriteLn bir dosyaya satır eklemek için kullanılmaktadır. *WriteLn* fonksiyonu, belirtilen dosyaya bir satır metin ekler ve otomatik olarak bir sonraki satıra geçmektedir.

Örnek kullanımı:

```
var
  yeniDosya: TextFile;
{
  AssignFile(yeniDosya, 'ornek.txt');
  ReWrite(yeniDosya);

  WriteLn(yeniDosya, 'Bu birinci satir');
```

```
  WriteLn(yeniDosya, 'Bu ikinci satir');

  CloseFile(yeniDosya);
}
```

Bu örnekte, *WriteLn* prosedürü, 'ornek.txt' adlı dosyasına, *WriteLn* ile iki satır metin eklemektedir. *CloseFile* prosedürü, dosyayı kapatmakta ve değişiklikleri uygulamaktadır.

16.1.5. CloseFile

Dosyayı kapatmak için kullanılan bir prosedürdür. Dosyanın kapatılması, üzerinde yapılan değişikliklerin uygulanmasını sağlayarak dosyanın artık kullanılmadığını belirtmektedir.

```
CloseFile(yeniDosya);
```

16.1.6. Append

Append prosedürü; dosyaya veri eklemek için kullanılmaktadır. *ReWrite* prosedürüyle oluşturulan dosyanın içeriği silinmez ve dosyanın sonuna eklemeye devem eder.

```
var
yeniDosya : TextFile;

{
  if (clFileExists('dosya.txt'))
 {
    AssignFile(yeniDosya, 'dosya.txt');
    Append (yeniDosya);

 WriteLn(yeniDosya, 'yeni');
    WriteLn(yeniDosya, 'World');
  }

    CloseFile(yeniDosya);
}
```

16.1.7. ReadLn

Dosyadan bir satır okumak için kullanılan bir prosedürdür. Bu prosedür, belirtilen dosyadan bir satırı okumaktadır.

```
var
yeniDosya : TextFile;

{
  if (clFileExists('dosya.txt'))
   {
```

```
    AssignFile(yeniDosya, 'dosya.txt');

    Reset(yeniDosya);
    while (not Eof(yeniDosya))
   {
     ReadLn(yeniDosya);
   }
  }
}
```

16.1.8. Reset

Bu prosedür, belirtilen dosyayı okuma veya yazma modunda açmak için kullanılmaktadır.

```
var
yeniDosya: TextFile;
{
  if (clFileExists('dosya.txt'))
  {
    AssignFile(yeniDosya, 'dosya.txt');

    Reset(yeniDosya);
     //Okuma veya yazma işlemi yapılabilir.
  }
}
```

Aşağıdaki örnekte Clomosy uygulamasının bulunduğu dosya dizininde aranan dosya var ise dosyanın üzerine yaz, yok ise dosyayı oluştur ve içerisine veri yaz işlemi yapılmaktadır.

```
var
txtDosya : TextFile;

{
  if (clFileExists('dosya.txt'))
  {
    AssignFile(txtDosya, 'dosya.txt');
    Append(txtDosya);
    WriteLn(txtDosya, 'İlk kayıt');
    WriteLn(txtDosya, 'İkinci kayıt');

  } else
  {
    AssignFile(txtDosya, 'dosya.txt');
    Rewrite(txtDosya);
    WriteLn(txtDosya, 'Yeni Kayıt 1');
  }
    CloseFile(txtDosya);

}
```

16.2. Özelleştirilmiş Dosya İşlemleri

TRObject programlama dilinde *TclStringList* sınıfı, metin verilerini içeren bir liste sağlamaktadır. Bu verileri kolayca bir dosyaya yazmak veya dosyadan okumak için kullanılmaktadır. Aşağıda *TclStringList* kullanarak bir dosya oluşturma işlemi gösterilmektedir.

Öncelikle TclStringList nesnesi oluşturulur. Bu nesne, metin verilerini içerecek olan listeyi temsil eder.

```
var
  strList: TclStringList;
{
  strList = Clomosy.StringListNew;
…
```

16.2.1. LoadFromFile

LoadFromFile yöntemi, bir dosyadan metin verilerini TclStringList nesnesine yüklemek için kullanılmaktadır. Bu yöntem kullanılırken dosyanın kontrolü yapılmalıdır. Dosya yolu istenilen yolda kayıtlı ise *TclStringList* nesnesine yüklenmelidir.

```
if clFileExists('MyFile.Txt',Clomosy.AppFilesPath)
  {
    ShowMessage('Dosya mevcut.');
    strList.LoadFromFile(Clomosy.AppFilesPath+'MyFile.Txt', 0);
    ShowMessage(strList.Text);
  }
```

16.2.2. SaveToFile

TclStringList nesnesindeki veriler, *SaveToFile* metodu kullanılarak bir dosyaya yazılır. Bu aşamada, dosyanın tam yolunu ve adını belirtmek önemlidir.

Dosya istenilen yol üzerinde var ise silinip üzerine yazılır.

```
var
  strList: TclStringList;
  dosyaStr: String;
{
   strList.Add('Satır 1: Merhaba dünya!');
   strList.Add('Satır 2: Bu bir örnek dosyadır.');
   strList.Add('Satır 3: Clomosy ile dosya oluşturuyoruz.');
   strList.SaveToFile(dosyaStr, 0);
}
```

Aşağıdaki örnekte dosya oluşturma, var olan dosyayı *TclStringList* nesnesine aktarma ve son olarak *TclJsonQuery* nesnesine aktarma işlemi yapılmaktadır.

```
var
  strList: TclStringList;
  dosyaStr: String;
  jsonSorgu: TclJsonQuery;

{
    strList = Clomosy.StringListNew;
  jsonSorgu = TclJsonQuery.Create(nil);
  dosyaStr = clPathCombine('MyFile.Txt', Clomosy.AppFilesPath);

  try

    if clFileExists('MyFile.Txt',Clomosy.AppFilesPath)
    {
      ShowMessage('Dosya mevcut.');
      strList.LoadFromFile(Clomosy.AppFilesPath+'MyFile.Txt', 0);
      ShowMessage(strList.Text);

    } else
    {
      ShowMessage('Dosya oluşturulup veri ekleniyor.');
      strList.Add('[{"ad":"Kader","yas":8},{"ad":"Mehmet","age":80}]');
      strList.SaveToFile(dosyaStr, 0);
    }

    if not(strList.Text == '')
    {
      jsonSorgu = Clomosy.ClDataSetFromJSON(strList.Text);
      ShowMessage(jsonSorgu.getJSONString);
    }
  finally

    strList.Free;
    jsonSorgu.Free;
  }
}
```

ile

Hayallerini Yaz

Clomosy, işletmelerin kendi ihtiyaçları doğrultusunda uygulama geliştirebilecekleri bir platformdur. Farklı platformlara uygun, kullanıcı dostu, geliştirilebilir, düzenlenebilir ve özel mobil uygulamalar geliştirebileceğiniz bulut tabanlı mobil uygulama geliştirme sistemidir. Platform bağımsızdır (Android, Ios, Windows). Kullanıcı verilerini istediği yerde kullanabilir.

Kullanıcı verileri Buluttadır. Kullanıcılar istedikleri platformu (Mobil cihaz veya Bilgisayar) seçerek projelerini kullanabilirler.

Clomosy'nin web arayüzünde fazla kodlama yapılmadan uygulamaya hakim olunmaktadır. Kimseye ihtiyaç duymadan kendi istekleri doğrultusunda istedikleri değişiklikleri yapabilirler.

Kolayca oluşturabileceğiniz mobil uygulamalar için oluşturduğunuz bilgiler bulut ile güvende ve tasarımlarınız her zaman yanınızda olacak. İster ticari ister kişisel, istediğiniz alanda benzersiz uygulamalar geliştirebilirsiniz.