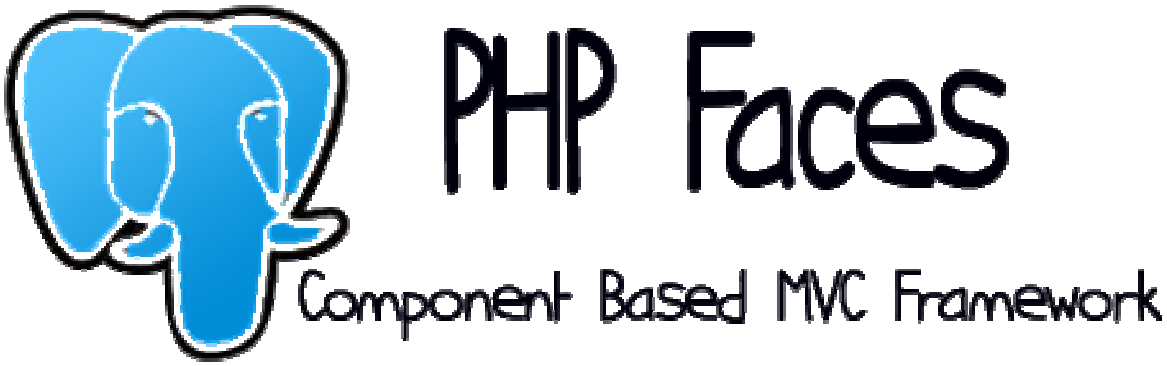


PHP Faces Framework
Uygulama Geliştirme Çatısı

Hazırlayan Hüseyin Bora ABACI

İçindekiler

Giriş

PHP Faces ile olay yönelimli programlama giriş kısmına kadar PHP Faces ile ilgli bazı temel tanımlamalar yer alacaktır Eğer bu konuda daha önceden bilgi sahibi iseniz bu sayfaları okumadan **PHP Faces ile olay yönelimli programlama giriş** başlığına bakabilirsiniz.

Öncelikle PHP Faces in ne olup ne olmadığına değinelim. PHP faces PHP için MVC tabanlı bir uygulama çatısıdır. Olay yönelimli ya da yönelimsiz olarak kullanılabilir. Kesinlikle php de var olan CodeIgniter, CakePHP, Zend gibi yapılarla karşılaştırılmaması gerekir. Bu yapıların hiç biri olay ve bileşen yönelimli değildir. PHP ile geliştirilen olay yönelimli başlıca frameworkler Borland firmasının geliştirdiği Delphi for PHP ve Prodo frameworkleridir.

PHP ile uygulama geliştirirken hissesilen bazı eksiklikler bu tarz bir framework geliştirme konusunda güdüleyici olmuştur. Java platformundaki popüler JTSL,JSF, Structs gibi frameworklerden ve Coldfusion dan etkilenilmiştir. PHP Faces JFS'nin basit bir PHP uyarlaması diyebilirim. içerisine JPA ya benzer birde ORM Katmanı bulunmakadır.

Olay yönelimli programlaya aşinansanız ve bu yöntemi benimsemiş seniz Web uygulamalarında PHP Faces iyi bir alternatif olabilir.

PHP Faces

View (Görünün için) **FDL**(Faces Definition Language) (Yüz Tanımlama Dili)

Model (Kalıp için) **FQL** (Faces) Query Language) (Yüz Sorgulama Dili)

İsimlerini verilen iki farklı dilinde içinde barındırır FDL XML kalıbını, FQL SQL kalıbını kullanır.

Belirgin özellikler

- MVC (Model-View-Controller)
- Component architecture
- UI components
- Event-driven programming
- AJAX
- Template Engine
- Custom Tags
- ORM (Object Relation Mapping)
- Validations(Doğrulama)

PHP Faces çalışma çatısını benzer çatılarla karşılaştırdığımda en önemli özelliğinin. Web programcılığına yeni bir kavram kazandıracağını düşündüğüm patterns (kalıplar) kavramıdır. Bunu bir Türk programcısı olarak Microsoft, Sun, IBM vs gibi firmaların mühendislerinden önce geliştirmiş olmak ve programlama adına yeni bir kavram kazandırmış olmaktan dolayı son derece mutluyuyuz. Kısaca patternler görünüm dosyanızda tanımladığınız ve sizi fazla html vs kod yazımından kurtaracak olan bir yöntemdir. İlerleyen sayfalarda pattern kullanımına deyinilecektir.

Bu bölümde son olarak PHP Faces i inceleyebileceğiniz adreslere yer veriyorum.

indirme sayfası

http://code.google.com/p/php-faces/
Örnekler için bu sayfayı ziyaret edin
http://phpfaces.webmahsulleri.com/

Kurulum ve Gereksinimler

Gereksinimler:

PHP 5 ve Üzeri
Aphache Rewide Module
Aphache CGI Module
ORM için PHP PDO eklentisini aktif hale getirin
Kurulum :

Sıkıştırılmış dosyayı çıkarın

PHPFaces klasörünü sunucunuza taşıyın

PHP modunda çalışmak.

PHP Faces normal PHP uygulamaları geliştir gibi MVC kalıbını kullanabilirsiniz bunun için fazladan bir ayarlama yapmanıza gerek yok. yapmanız gereken taban URL adresini belirleyip PHP faces ı kullanmak istediğiniz php dosyanıza phpfaces/system/PHPFaces.php dosyasını dahil etmek. Dillerseniz bir yapılandırma dosyası hazırlayıp bu dosyayı dahil edebilirsiniz böyle çalışmak daha mantıklı olacaktır.

Örnek

```
<?php
define("BASE_URL","http://localhost/phpfaces/");
require_once ("phpfaces/system/PHPFaces.php");
import("phpf.controllers.facete");
class Sayfa extends Facete {
    function Sayfa() {
        parent::Facete();
        $this->render("gorunum/view.html");
    }

    function dugmeClicked($e) {
        $e->getComponent()->text="bana tıklandı";
    }
}
$ Dispatcher::dispatchclass("Page");
?>
```

Dizin yapısı ile çalışmak
Ana dizinde bulunan index.php dosyasını bir editor ile açın ve düzenleyin

Dispatcher::dispatch parametrelerini sisteminize göre ayarlayın.

Dispatcher::dispatch("applications".DS."seninuygulama","SeninKontrolcu","http://seninhost.com/");

Dispatcher ın parametreleri

1- geçerli uygulama dizini

2- uygulamanın başlayacağı controller ismi

3-base url (taban adres) PHP Faces in çalıştığı adrestir

application/config.php dosyasını açın ve kendi sistem özelliklerinize göre düzenleyin.

define(“DEFAULT_APPLICATION”,“myapp”); //birincil uygulama dizini

define(“DB_CONNECTION_STRING”,“mysql:host=localhost;dbname=mydb”);// ORM için bağlantı dizesi

define(“DB_USER”,“username”);
define(“DB_PASS”,“2w34r5t”);
define(“BASE_URL”,“http://seninsunucun/”);//bu tanım eğer xxx.phpf gibi uzantılı dosyaları kulanmak isterseniz gerekli bunun için bazı .htaccess ayarlamaları yapmanız gerekiyor.

PHP faces güzel görünen yol sitili URL biçimini destekler bunun için. htaccess dosyanızda aşağıdaki satırlar bulunmalı.

RewriteEngine on
RewriteCond $1 !^(index\.php|images|themes|css|js|video_files|robots\.txt|favicon\.ico)
RewriteCond %{REQUEST_FILENAME} !-f

RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^(.*)$|index.php$ index.php/$1 [L]

PHPF uzantılı dosyaları kullanabilmek için Aphache üzerinde CGI modülü açık olmalı ve htaccess dosyanızda aşağıdakine benzer satırlar bulunmalı.

Options +ExecCGI
AddType application/x-httpd-php .phpf s
AddHandler application/x-httpd-phpf .phpf
Action application/x-httpd-phpf www.seninsunucun.com/system/phpf.php

yukarıda yaptığımız işlemler sayesinde suncumuza .phpf gibi bir istek geldiğinde bu PHP Faces framework bunu algılar .phpf uzantılı dosyalar view(görünüm) dosyalarıdır. PHP Faces böyle bir istek durumunda. phpf ile aynı isimdeki .php controller (kontrolcü) yü harekete geçirir. artık yönetim controller da dır.

Faces MVC (Model View Controller)

Model-view-controller, yazılım mühendisliği'nde kullanılan bir "mimari desen"dir. Kullanıcıya yüklü miktarda verinin sunulduğu karmaşık uygulamalarda veri ve gösterimin soyutlanması esasına dayanır. Böylelikle veriler (model) ve kullanıcı arayüzü (view) birbirini etkilemeden düzenlenebilir. Model-view-controller, bunu controller adı verilen ara bileşenle, veri gösterimi ve kullanıcı etkileşiminden, veri erişimi ve iş mantığını çıkarma suretiyle çözmektedir.

M (Model), İş Mantığını ve data işleme süreçlerini yürütür. **C** (Controller) tarafından gönderilen emirlere göre hareket eder. Bilgi işleme sürecinden sonra datayı C'ye, diğer modellere veya doğrudan **V** (View)'ye gönderir.

PHP Faces Model katmanında **PDO** ve ORM Kullanır

V (View) son kullanıcıya gösterilecek olan datanın sunumu ile ilgilenir. V, bu bilgiyi C veya **M**'den alır, aynı zamanda son kullanıcıdan gelen talepleri **C**'ye iletir.

C (**Controlle**r)
C ise sistemin ana kısmıdır. Gelen talepleri kontrol eder ve sistemin diğer elemanlarının (M,V) bilgiyi uygun şekilde alıp, göndermelerini sağlar.

PHP Faces içerisinde üç farklı Controller barındırır bunlar

1. ActionController
2. FacesController
3. Facet

ActionController sıradan olay yönelimli olamayan bir yapıdadır Kullanıcıdan bilgi girişlerinin istenmediği durumlarda kullanışlıdır.

FacesController olay yönelimlidir. Faces bileşenleri ile birlikte kullanılır.

Facet FacesController dan genişletilmiş FacesController u basitleştirmiştir.

PHP Faces i diğerler MCV Çatılarından ayıran özelliği ise Controller ve View arasında bileşenlerin ve kod işleyici basit anlamda yorumlayıcıların bulunmasıdır.
Bir view dosyası kontrolcünün render metodu ile istemci tarafındaki browser a bir çıktı oluşturulur. Render metodu kendisine parametre olarak verilen dosyayı basit bir yorumlama işleminden geçirir.

PHP Faces MVC için dizin yapısı aşağıdaki gibidir.

AppPath:
/phpfaces : Framework folder
/applications: uygulamalar

app_bir /controllers
/views
/models
app_two /controllers
/views
/models
.htaccess

Faces ile ufak bir başlangıç

Bu örneğin dizin yapısında olduğunu hatırlatırım Eğer dizin yapısı kullanmak istemiyorsanız normal php dosyaları şeklinde de kullanılabilir bu durumda Kurulum sayfasında anlatıldığı gibidir. Daha öncede bahsettiğimiz gibi MVC uygulamalarının omurgası Controller kısmıdır. Bir kontrolcü oluşturacağınız zaman kontrolcünüzü ilgili kontrolcü sınıfından genişletirsiniz.

Evet her şey hazırsa yani kurulum işlemleri başarı ile sonuçlamışsanız ilk PHP Faces denememizi yapabiliriz

Öncelikle
applications/uygulamaadi/controller/

applications/uygulamaadi/controller/test.php

Dizinine gelin burada test.php gibi bir isimde Controller dosyamızı oluşturun.

Örnek kontrolcümüzün içeriği

```
<?php
import("phpf.controllers.facescontroller");
class Test extends FacesController {
    function Test() {
        parent::FacesControler();
        echo "Test kontrolcüsü başlatıldı";
    }
}
?>
```

Şimdi tarayıcımızda adres satırına localhost da çalıştığınızı düşünerek

http://localhost/phpFacesdizini/ugulamaadi/test

Yazıyoruz eğer her şey yolunda gitmişse ” Test kontrolcüsü başlatıldı” mesajını alıyoruz.

Şimdi

applications/uygulamaadi/views/

Dizinine gelin burada test.phpf gibi bir isimde basit bir view oluşturalım.

faces kodlarımızı <faces> </faces> arasına yerleştiriyoruz FDL anlatırken bunlara derinlemesine değineceğiz

```
<faces>
<b>#{$this.mesaj}</b>
</faces>
```

Controller sınıfımızı da aşağıdaki gibi düzenleyelim Kontrolcünün render metodu bir view dosyasını işleyerek istemci tarafına bir çıktı oluşturur.

```
<?php
import("phpf.controllers.facescontroller");
class Test extends FacesController {
    protected $mesaj="Test kontrolcüsü başlatıldı";
    function Test() {
        parent::FacesControler();
        $this->render("test.phpf");
    }
}
?>
```

Yeniden

http://localhost/phpFacesdizini/ugulamaadi/test

Yazıyoruz eğer her şey yolunda gitmişse ” Test kontrolcüsü başlatıldı” mesajını alıyoruz.

İmport Fonksiyonu

İmport fonksiyonu kullanmanız için gerekli olan php sınıflarını dosyanıza dâhil eder. Php nin require_once(string filename) ; fonksiyonunu işletir. Aynı dizinde bulunan tüm dosyaları dahil etmek için son karakter yıldız * karakteridir.

import("phpf.controllers.facescontroller");

Yukarıdaki bu satır ile phpf/controllers/facescontroller.php dosyası dahil edilmektedir.

import("phpf.ui.*");

Yukarıdaki bu satır ile phpf/ui/ dizini içerisindeki tüm php dosyaları dahil edilmektedir. İmport fonksiyonunun ikinci parametresi true ya da false değeri alır bu parametrenin true verilmesi durumunda import fonksiyonu ilgili dosyalara varsayılan uygulama dizini içerisinde arayacaktır.

import("models.*");

Yukarıdaki bu satır ile varsayılan uygulama dizininde models klasörünün altındaki tüm php dosyaları import edilir.
Örnekler

```
<?php
import("phpf.controllers.facete");
import("model.*",true);
import("io.uri");
?>
```

Faces Views

Görünüm'ü tamamen diğer kodlardan ayırabilirsiniz Faces View ler XML notasyonuna benzer. PHP Faces yapısı Java Server Faces,Coldfusion dan bazı özellikler almış kendine has bazı özelliklerde eklemiştir.

Faces Render ına tabi olacak bölümler **<faces>** **</faces>** etiketleri arasına yazılır.<faces> </faces> etiketleri arasında HTML ,CSS,XML PHP vb.. dilleride kullanabilirsiniz.

Faces Etiketleri üç e ayrılır bunlar

- Renderer direktif etiketleri
- Static etiketler
- Bileşen Etiketleri

Renderer direktiflerinin tamamı @ işareti ile başlar

```
<@import prefix="f" taglib="phpf.ui.*"/>
```

bu direktif renderer a phpf.ui içerisindeki tüm sınıfları dahil etmesini söyler ve f isim uzayı ile ilişkilendirir.

```
<@import prefix="c" taglib="phpf.core" type="static"/>
```

bu direktif renderer a phpf.core sınıfını dahil etmesini söyler ve c isim uzayı ile ilişkilendirir. static ibaresi bu sınıftaki metotların static olarak çağrılması gerektiğini bildirir.

Örnek

```
<faces>
<@import prefix="c" taglib="phpf.core" type="static"/>
<c:out value="Merhaba Faces"/>
</faces>
```

Başka bir örnek

```
<faces>
<@import prefix="c" taglib="phpf.core" type="static"/>
<@import prefix="f" taglib="phpf.ui.*"/>
<c:if test="1<2">
<c:out value =" bir ikiden kucuktur"/>
</c:if>
<f:button name="button" text="tıkla"/>
</faces>
```

Nitelik tanımlamalarında tek tırnak ‘ ‘ ya da boşluk kullanmayın çift tırnak ” ” karakterlerini kullanın ve çift tırnak içerisinde de ‘ tek tırnak işareti kullanmayın

Örnekler

```
<faces>
<@import prefix="c" taglib="phpf.core" type="static"/>
<c:out value=deneme/> yanlış hata alırsınız
<c:out value='deneme'/> yanlış hata alırsınız
<c:out value="doğru"/>çalışır
<c:out value="yanlış’ın"/>çalışmaz tek tırnak var
</faces>
```

Faces ifadeleri

PHP faces ifadeleri view(Görünüm) içerisinde kullanılan ifadelerdir. Bu ifadeler bir çıktı ya da bir bileşene göndermede bulunur
İfadeler #{ ve } aralığına yazılır.

Örnekler

```
<b>
#{$varible}
</b>
```

Yukarıdaki örnek $varible içeriğini yazdırır

```
#{1+2}
#{$i+$j}
#{$i*$j}
```

Örnek HTML içerisinde ifade kullanımı

```
<faces>
<@import prefix="c" taglib="phpf.core" type="static"/>
<input type = "text" name="text" value="#{$varible}"/>
<c:for var="$i" begin="0" to="10" step="1">
<input type = "text" name="text#{$i}" value="#{$i}"/>
</c:for>
</faces>
```

@import direktif etiketi

Direktif etiketleri görünüm (view) dosyalarında tanımlama işlemleri için kullanılan faces renderer içerisinde bulunan etiketlerdir her bir direktif @ işareti ile başlar ve her direktifin spesifik nitelikleri vardır.
İmport direktifi taglib niteliği ile belirtilen php faces bileşen dosyalarını dâhil eder ve prefix niteliği ile belirlenen ön ek ile ilişkilendirir. Type nitelği ise kullanılacak kütüphanenin çalışma şekliyle ilgilidir eğer PHP sınıf kütüphanesi durağan(statik) metotlar içeriyorsa bu niteliğe true değeri verilmelidir. Core.php sınıfı statik bir kütüphanedir.

- taglib: import edilecek olan dizin ve ya php dosyası
- prefix : etiketler ile kullanılacak ön isim
- type: bileşen kütüphanesinin tipi

type niteliği sadece statik sınıflarda kullanılır ve PHP Faces da şuan tek statik kütüphane core.php dir

Örnek

```
<faces>
<@import taglib="phpf.core" prefix="c"  type="static"/>
<c:out value="Merhaba Faces"/>
</faces>
```

Yukarıda phpf/core.php dizini içerisindeki Core statik sınıfı import edilmiş ve “c” ön eki ile ilişkilendirtmiştir. Core ile sınıfı ile ilişkili etiketler “c:” başaklayacaktır.
<c:out value="Merhaba Faces"/> satırı ile core sınıfının out metodu çalıştırılır ve tarayıcıya Merhaba Faces yazısı yazdırılır.

Başka bir örnek

```
<faces>
<@import prefix="c" taglib="phpf.core" type="static"/>
<@import prefix="f" taglib="phpf.ui.*"/>
<c:if test="1<2">
<c:out value =" bir ikiden kucuktur"/>
</c:if>
<f:button name="button" text="tıkla"/>
</faces>
```

Yukarıda da c ile yine core sınıfı ilişkilendirilmiş ardından phpf/ui dizinindeki tüm sınıflarla “f “ ön eki ilişkilendirilmiştir.

Başka örnek bir daha

```
<faces>
<@import prefix="core" taglib="phpf.core" type="static"/>
<@import prefix="face" taglib="phpf.ui.*"/>
<@import prefix="widget" taglib="phpf.ui.widget.djbutton"/>
<core:out value =" Hello World"/>
<face:button name="button" text="Click me"/>
<widget:djbutton name="djbutton" text="Widget button"/>
</faces>
```

Yukarıda da core ön eki ile core sınıfı ilişkilendirilmiş ardından phpf/ui dizinindeki tüm sınıflarla “face “ ön eki ilişkilendirilmiş ve “widget”. Ön eki ile de phpf/ui/widget/djbutton.php içerisindeki DJButton sınıfı ilişkilendirilmiştir.

Bileşenler tanımları ve kullanımları

PHP Faces MVC içerisinde bileşeler görünüm (View) içerisinde tanımlanır. Bileşenler birer PHP sınıflarıdır.
İmport direktifinde belirtilen ön ek isminden sonra: iki nokta üst üste işareti bileşen adi ve nitelikleri şeklinde tanımlanır.

Örnek

```
<faces>
<@import prefix="f" taglib="phpf.ui.button"/>
<f:button name="dugme" text="tıkla"/>
</faces>
```

Yukarıdaki satırlarda button sınıfından dugme adında yeni bir nesne oluşturulmakta ve text niteliğine tıkla değeri verilmektedir. Text niteliği oluşturulan butonun üzerinde yazılı olacak yazıdır.
Her bir bileşen etiketinin name ya da id nitelikleri boş bırakılmamalıdır. PHP içerisinden nesnelere erişirken bu isimler ile erişilecektir.
Yukarıdaki örneğe göre php kontrolcü sınıfımız da dugme isminde bir button nesnesine sahiptir.
PHP içerisinden dugme nesnesine aşağıdaki gibi erişilir.

```
$this->dugme->text="yeni yazı";
```

PHP Faces ile olay yönelimli programlama giriş

PHP faces içerisinde olay yönelimli programlamayı destekleyen iki adet kontrolcü sınıfı bulundurur bunlar FacesController ve Facete sınıflarıdır. Her iki kontrolcüde de bir view dosyasını işlemek istediğinizde render metoduna işlemek istediğiniz view dosyasının adını parametre olarak yazarsınız

Facete

Facete sınıfı FacesController sınıfı genişletir ve ActionListener,MouseListener,ValueChangedListener arayüzlerini (interface) uygulayan bir sınıftır.
Facete ile Vbasic, Delphi deki olay yönelimine yakın bir şekilde programlanmıştır.
Bir facete kontrolcüsü oluşturmak için sınıfınızı Facete sınıfından genişletirsiniz ve sınıfınız ile aynı isimdeki kurucu metodu hazırlarsınız.

Olay yakalama (Event Handling) üye metotlar ile sağlanır. Olayı yakalayacak metodun ismi önemlidir. Olay yakalamada metot isimleri nesne adı olay adı şeklindedir.

Örneğin Ornek adında bir Facete sınıfımız ve dugme adında gorunum dosyasında tanımını yaptığımız bir buton olsun butona tıklanıldığında buton üzerindeki yazıyı değiştirmek istiyoruz.

Görünüm Dosyamız

```
<faces>
      <@import taglib="phpf.ui.button" prefix="f"/>
      <f:button name="dugme" text="Tıkla" onclick="actionevent"/>
</faces>
```

Facete Kontrolcümüz

```
<?php
import("phpf.controllers.facete");
class Ornek extends Facete {
    public function Ornek() {
        parent::Facete();
        $this->render("view.html");
    }
    function dugmeClicked($evt) {
        $this->dugme->text="Tıklatıldım";
    }
}
?>
```

Kullanıcı tarafından dugme butonuna tıklandığında sayfa post edilir ve dugmeClicked metodu çalıştırılır.
Eğer sayfayı yenilemeden AJAX ile verilerin gönderilmesini isterseniz uygulamanızda birkaç değişiklik gereklidir.
Görünümde buton nesnesini tanımlarken kullandığımız onclick olayına ajaxevent değerini verdiğimizde bu nesnenin ajax ile sunucuya gönderileceğini söylemiş olursunuz.
dugmeClicked metodu sonunda da $this->AjaxResponse(); metodunu çalıştırmamız gerekmektedir. Bu metot tarayıcı tarafına JSON verisi aktarır.

Örneğimizin AJAX ile çalışır hali aşağıdadır.

Görünüm Dosyamız

```
<faces>
      <@import taglib="phpf.ui.button" prefix="f"/>
      <f:button name="dugme" text="Tıkla" onclick="ajaxevent"/>
</faces>
```

Facete Kontrolcümüz

```
<?php
import("phpf.controllers.facete");
class Ornek extends Facete {
    public function Ornek() {
        parent::Facete();
        $this->render("view.html");
    }
    function dugmeClicked($evt) {
        $this->dugme->text="Tıklatıldım";
        $this->AjaxResponse();
    }
}
?>
```

FacesController

Olay yönelimli diğer bir yaklaşım ise FacesController kullanmaktır. PHP Faces in olay mekanizması içereninde üç önemli kavramı anlamanızda yarar vardır. PHP Faces olay yakalama kalıpları Java tarzındadır. eğer java swing ile daha önce uygulama geliştirmişseniz bu tanımlar size yabancı gelmeyecektir.

- Listener : Dinleyiciler
- Event: Olaylar
- Component : bileşennler

Listener bir ara yüz interface tanımlar Controller sınıfı da dinlemeye alacağınız bu ara yüzleri uygularsınız.

Eventlar gerçekleşen olay ile ilgili nesnelerdir. Bunlar framework tarafından oluşturulur ve olay ile ilgili metot parametrelerine geçirilir.

Component genellikle olaylar bir bileşen tarafından tetiklenir.

PHP Faces da şuan bulunan listenerlar

- ActionListener: tıklama olayı için kullanılır
- MouseListener : mouse olayları için kullanılır
- ValueChangedListener: değerler değiştiğinde kullanılır

```
<?php
interface ActionListener extends FacesListener {
/**
 * @param ActionEvent $event
 */
    public function actionPerformed(ActionEvent $event);
}?>
```

PHP Faces1.0 da şuan bulunan olaylar nesneleri

- ActionEvent: bir tıklama olayı gerçekleştiğinde
- MoueseEvent: bir mouse olayı gerçekleştiğinde
- ValueChangedEvent: değişim olayı gerçekleştiğinde

controler sınıfımız listener ile ilgili metodunu çağırır ve listener uygulayan sınıfı olay için kayıt eder.

PHP faces olaylar için temel iki post yöntemi kullanır ilk yöntem post back ve ikinci yöntem AJAX tır.

Olayların gerçekleşmesi için view(görünüm) tarfında bileşen tanımlamaları yapılırken bildirilmesi gerekir.

Bileşenin post back yapmasını istiyorsanız bileşen tanımında ilgili olay nitelğine “actionevent”

bileşenin bir AJAX çağrısı yapmasını istiyorsanız bileşen tanımında ilgili olay nitelğine “ajaxevent” değerlerinden birini verirsiniz.

Örnek bir bileşen ve olay tanımlaması

```
<faces>
<@import prefix="face" taglib="phpf.ui.button" />
<face:button name="button1" text="Tıkla" onclick="ajaxevent"/>
</faces>
```

Yukarıdaki satırlar ile bir button sınıfından button1 adında yeni bir örneğini oluşturduk. button1 nesnesine controller içinden erişebiliriz

Yukarıdaki viewımız için örnek controller

```
<?php
import("phpf.controllers.facescontroller");
import("phpf.events.actionevent");
import("phpf.listeners.actionlistener");
class Sample extends FacesController implements ActionListener {
    function Sample() {
        parent::FacesController();
        $this->addActionListener($this);
        $this->render("view.phpf");
    }
    public function actionPerformed(ActionEvent $evt) {
        $this->button1->setText("Bana tıkladın");
        $this->AjaxResponse();
    }
}
?>
```

Yukarıdaki satırları özetlemek gerekirse

class Sample extends FacesController implements ActionListener

ActionListener dinleyicisi uygulanıyor.

$this->addActionListener($this);

dinleyici nesne kayıtediliyor.

$this->render(”view.phpf”);

view dosyamız render ediliyor.

İstemci tarayıcı tarafında oluşturduğumuz buttona tıklandığında ActionEvent olayı meydana geliyor ve actionPerformed metodu çalıştırılıyor.

public function actionPerformed(ActionEvent $evt){

$this->button1->setText(”Bana tıkladın”);
button nesnesinin text niteliği bana tıkladın olarak değiştiriliyor.

$this->AjaxResponse();
İstemci tarafına internet tarayıcısına JSON verisi gönderiliyor. Tarayıcı tarafındaki Ajax güncelleme kodları çalıştırılıyor.

FacesController ve Facete nin üye metotları aşağıdaki gibidir.

- **render** (Bu metor renderer a bir faces dosyasını yorumlamasını söyler)
- **prerender(**Render işlemi başlatılıp view dosyası işletilip çıktının henüz oluştutulmadığı zaman işletilir **controlle**r içerisinde overide edilirek kullanılır**)**
- **renderend(**render işleminin tamınlandığında çalıştırılır **controller** içerisinde overide edilirek kullanılır**)**
- **append** (view a bir parametre gönderir)
- **load** (parametre olarak verilen string ifadenden sınıf adından yeni bir nesne çevirir)
- **interrupt** (render işlemini keser)
- **addListener**($name,FacesListener $listener)
- **addActionListener** (bir listener arayuzunu(interface) ActionEvent olayı için kayıteder parametresi ActionListener dır)
- **addMouseListener** (bir listener arayuzunu mouse events olayları içinkayıt eder Parametresi MouseListener)
- **addValueChangedListener**(bir listener arayuzunu ValueChangedEvent olayı için kayıt eder. Parametresi ValueChangedListener)
- **getComponents** (contoller ile ilişkili tüm bileşenleri verir)
- **AjaxResponse** (render işlemini askıya alır ve istemci tarafına json verisi gönderir)
- **setValidCallBack**(boolean (doğrulama işlemlerinde olayların işletilmesi gerekiryorsa kullanılır bu durumda bu metoda true değeri verilir))
- boolean **isValid()**(doğrulama işleminin sonucunda bir hata varsa false yoksa true değer döndürür)

prerender ve renderend metotları Controller içerisinde override edilerek kullanılır
Örnek

```
<?php
import("phpf.controllers.facete");
class Render extends Facete {
    public function Render() {
        parent::Facete();
        $this->render("dosya.html");
    }
    public function prerender() {
        echo "Render işlemi başlatıldı nesneler oluşturuldu";

    }
     function renderend(){
     echo "Render işlemi sona erdi sayfa oluşturuldu";
    }
  }
?>
```

PHP Faces Model

PHP faces model katmanında ORM(Object relation mapping) kullanır bu bölüm ayrı bir dokuman olarak anlatılıcaktır.

PHP Faces Core Sınıfı

Core sınıfı statik bir etiket kütüphanesidir. Statik kütüphaneler ile Bileşen kütüphaneleri arasındaki farklılık Bileşenlerin bileşen sınıfından yeni birer nesne olarak üretilmesi ve kontrolcü birer üyeleri olması. Statik sınıflar ise statik metotlardan oluşan renderer'a çıktı üreten sınıflardır. Bu etiketler name ve id değeri almazlar. Core sınıfı bazı temel işlemlerin bulunduğu statik etiket sınıfıdır. Core sınıfı phpf dizini içerisinde bulunur. Bu bölümde uygulanan örneklerde bir kontrolcü oluşturmayı ve örneklerin render edilmesi gerektiğini unutmayın.

```
<@import prefix="core" taglib="phpf.core" type="static"/>
```

Yukarıdaki gibi görünüm dosyasına import edilebilir.
Core sınıfı içerinde aşağıdaki etiketler bulunur.

- out
- if
- else
- elseif
- for
- foreach

Out Etiketi

Çıktı için kullanılır. Çıktıya gönderilecek veri bu etiketin value niteliği ile belirtilir

Örnek Out

```
<faces>
<@import prefix="c" taglib="phpf.core" type="static"/>
<c:out value="Merhaba dünya"/>
<c:out value="#{birdeğişken}"/>
</faces>
```

if etiketi

İf etiketi temel karşılaştırma işlemlerinde kullanılır if etiketinin test adında bir niteliği bulunur ve bu niteliğe karşılaştırma ifadesi yazılır. Açılan her if etiketi kapatılmak zorundadır.

```
<c:if test="#{ifade}">
Test edilen ifadenin doğruluğu durumda bu satırlar işlenir.
</c:if>
```

Örnek İf

```
<faces>
<@import prefix="c" taglib="phpf.core" type="static"/>
<c:if test="#{$name==bora}">
<c:out value="#{$name}"/>
</c:if>
</faces>
```

İf için baş kabir örnek

```
<faces>
<@import prefix="c" taglib="phpf.core" type="static"/>
<c:if test="#{$i>0  AND $i<10}">
<c:out value="sayı  10 ile  1 arasinda"/>
</c:if>
</faces>
```

İf için başka bir örnek

```
<faces>
<@import prefix="c" taglib="phpf.core" type="static"/>
<c:if test="#{$name!=huseyin}">
<strong>
<c:out value="Bu benim adım değil !"/>
</strong>
</c:if>
</faces>
```

Else Etiketi

Else etiketi if etiketi ile birlikte kullanılır. if ile kullanılan ifaden farklı olması durumunda geçerlidir. PHP deki else kullanımı gibidir.

Örnek else

```
<faces>
    <@import prefix="c" taglib="phpf.core" type="static"/>
    <c:if test="$name==bora">
        <c:out value="#{$name}"/>
        <c:else>
            <c:out value="Başka bir isim  #{$name}"/>
        </c:else>
    </c:if>
    <faces>
```

Yukarıdaki örnekte $name değişkeni test edilir eğer değişken değeri bora ise bu değişken out etiketi ile yazdırılır. Değişken değeri farklı bir değerse “Başka bir isim ve $name değişkeninin değeri yazdırılır.

Çift ve tek sayıları ayırt eden bir örnek

```
        <faces>
            <@import prefix="c" taglib="phpf.core" type="static"/>
            <c:for var="$i" begin="1" to="12" step="1">
                <c:if test="($i%2)==1">
```

```
                <c:out value="#{$i}"/> tek sayı<br />
                <c:else>
                    <c:out value="#{$i}"/> çift sayı<br />
                </c:else>
            </c:if>
        </c:for>
        <faces>
```

Else if Etiketi

elseif etiketi if etiketi ile birlikte kullanılır. if ile kullanılan ifadenin yanlış olması durumunda elseif etiketinin test niteliğindeki ifadenin doğrulu karşılaştırılır.

Örnek elseif

```
<@import prefix="c" taglib="phpf.core" type="static"/>
<faces>
    <@import prefix="c" taglib="phpf.core" type="static"/>
    <c:if test="#{$name==bora}">
        <c:out value="#{$name}"/>
        <c:elseif test="#{$name==huseyin}">
            <c:out value="isim huseyin"/>
        </c:elseif>
    </c:if>
</faces>
```

Örnek if elseif else

```
<faces>
    <@import prefix="c" taglib="phpf.core" type="static"/>
    <c:if test="#{$name==bora}">
        <c:out value="isim bora"/>
        <c:elseif test="#{$name==huseyin}">
            <c:out value="isim huseyin"/>
        </c:elseif>
        <c:else>
            <c:out value="tanımsız bir isim"/>
        </c:else>
    </c:if>
</faces>
```

For etiketi

for etiketi döngü oluşturmak için kullanılır. dört niteliği bulunmaktadır.

- var : değişken
- begin : başlama değeri bir tam sayı
- to : bitiş değeri bir tam sayı
- step : artış miktarı

var ile belirten değişkeni begin ile belirtilen başlangıç değerinden başlayarak to ile belirtilen miktara ulaşıncaya kadar step ile belirtilen miktar kadar artırır veya azaltılır.

Örnek 1 den 10 kadar olan sayıları göster.

```
<faces>
```

```
    <@import prefix="c" taglib="phpf.core" type="static"/>
    <c:for var="$i" begin="1" to="10" step="1">
        <c:out value="#{$i}"/>
    </c:for>
</faces>
```

Çıktısı şöyle olur 1,2,3,4,5,6,7,9,10

Örnek ikişer artış çift sayılar

```
<faces>
    <@import prefix="c" taglib="phpf.core" type="static"/>
    <c:for var="$i" begin="0" to="10" step="2">
        <c:out value="#{$i}"/>
    </c:for>
</faces>
```

Çıktısı şöyle olur 0,2,4,6,8,10

Örnek geriye doğru for

```
<faces>
    <@import prefix="c" taglib="phpf.core" type="static"/>
    <c:for var="$i" begin="10" to="0" step="1">
        <c:out value="#{$i}"/>,
    </c:for>
</faces>
```

Çıktısı şöyle olur 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0

Örnek iç içe for kullanımı

```
<faces>
    <@import prefix="c" taglib="phpf.core" type="static"/>
    <c:for var="$i" begin="0" to="10" step="1">
        <br>
        <c:out value="i = #{$i}"/>
        <c:for var="$j" begin="$i" to="10" step="2">
            <c:out value="j = #{$j}"/>,
        </c:for>
    </c:for>
</faces>
```

Çıktısı şöyle olur

i = 0, j = 0, j = 2, j = 4, j = 6, j = 8, j = 10
i = 1, j = 1, j = 3, j = 5, j = 7, j = 9
i = 2, j = 2, j = 4, j = 6, j = 8, j = 10
i = 3, j = 3, j = 5, j = 7, j = 9
i = 4, j = 4, j = 6, j = 8, j = 10
i = 5, j = 5, j = 7, j = 9
i = 6, j = 6, j = 8, j = 10
i = 7, j = 7, j = 9
i = 8, j = 8, j = 10
i = 9, j = 9

i = 10, j = 10

Foreach etiketi

For etiketine benzer bir şekilde çalışır diziler ve nesneler üzerinde etkilidir. var ve item isimlerinde iki niteliği vardır

- var: iterasyon yapılacak nesne ya da dizi
- item : işlem sırasındaki geçerli nesne

Örnek foreach

```
<?php class Person {
    public $id;
    public $name;
    public $phones =array();
}
?>
```

Yukarıdaki person sınıfından oluşturulan bir diziyi dolaşan foreach örneği

örnekteki $this.list Person sınıfı dizisidir.

```
<faces>
    <@import prefix="c" taglib="phpf.core" type="static"/>
    <c:foreach var="$this.list" item="$item">
        <br/>
        Name :<c:out value="#{$item.name}"/>
        Id :<c:out value="#{$item.id}"/>
    </c:foreach>
<faces>
```

Örnek iç içe foreach etiketleri

```
<faces>
    <@import prefix="c" taglib="phpf.core" type="static"/>
    <c:foreach var="$this.list" item="$item">
        <br/>
        Name :<c:out value="#{$item.name}"/>
        Id :<c:out value="#{$item.id}"/>
        <c:foreach var="$item.phones" item="$phone">
            Phone :<c:out value="#{$phone}"/>
        </c:foreach>
    </c:foreach>
<faces>
```

Bileşenler

PHP Faces içerisinde pek çok bileşen bulunmaktadır. kendi bileşenlerinizi de geliştirebilirsiniz. bu konuya ilerleyen sayfalarda değiniceyiz.

PHP Faces içerisinde DOJO Java Script frameworkunu barındırmaktadır. Dojo hakkında detaylı bilgiyi http://www.dojotoolkit.org/ adresinde bulabilirsiniz.

Şuan için PHP Faces ile birlikte ui ve widget adı altında 2 adet bileşen kütüphanesi bulunmaktadır. ui altında standart HTML form bileşenleri ui.widget altında DOJO dijit javascript tabanlı bileşenler bulunmaktadır.

Bileşenler için kullanım örneklerini http://www.webmahsulleri.com/faces/doc/index.phpf adresindeki HTML UI ve Widgets UI başlıkları altında bulabilirsiniz.

UI HTML Bileşenleri

- Button
- TextBox
- Label
- ComBox
- CheckBox
- Image
- Hidden
- Radio
- Form
- Grid
- TextArea

Button

Button bileşeni bir html buton etiketi üretir genellikle ***onclick*** olayı ile birlikte kullanılır buton bileşeninin text niteliği butonun üzerindeki yazı anlamına gelmektedir. HTML input text e ait tüm nitelikleri destekler.

Örnek view içerisinde Buton tanımlaması

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
    <title>Ornek Button</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<faces>
    <@import taglib="phpf.ui.button" prefix="f"/>
    <f:button name="dugme" text="Dugme" onclick="ajaxevent"/>
</faces>
</body>
</html>
```

Örneğimizin Kontrolcü tarafı

```
<?php
import("phpf.controllers.facete");
class Ornek extends Facete {
    public function Ornek() {
        parent::Facete();
        $this->render("ornek.html");
    }

    protected function dugmeClicked($olay) {
        $this->dugme->settext("Bana Tıkladınız");
        $this->AjaxResponse();
    }
}
?>
```

Dugme bileşeninin tıklanmadan önceki görünümü

Dugme bileşeninin tıklandıktan sonraki görünümü

Bileşenler istenirse kontrolcü içerisinde de oluşturula bilir. Bu bileşenler ***protected*** ve ya ***public*** üyeler olmalıdır. Bileşenlerin kurucu metotları bir kontrolcü sınıfı parametresi kabul eder.

```
$this->dugme = new Button($this);
```

Örnek Dugme bileşenin kontrolcü tarafında oluşturulması

```
<?php
import("phpf.controllers.facete");
import("phpf.ui.*");
class Ornek extends Facete {
    protected $dugme =null;
    public function Ornek() {
        parent::Facete();
        $this->dugme = new Button($this);
        $this->render("ornek.html");
    }

    protected function dugmeClicked($olay) {
        $this->dugme->settext("Bana Tıkladınız");
        $this->AjaxResponse();
    }
}
?>
```

Form

Form bileşeni bir HTML formu tanımlar formu post etmek için Button nesnesi kullanılabilir. PHP Faces a benzer olay yönelimi çoğu framework otomatik bir HTML form'u oluşturur ve tüm bileşenleri bu form'un içinde saklar. PHP Faces bu yöntemi kullanıcı tanımlı hale getirmiştir. Form içerisindeki tüm HTML tüm bileşenleri sunucuya gönderir eder. Form dışında kalan bileşenler ise sadece kendilerini sunucuya gönderirler. Form kullanımı HTML benzer bir şekildedir. Formu post etmek istediğiniz Button ya da farklı bir bileşenin ***forname*** niteliğine post etmek istediğiniz form bileşeninin adını yazarsınız.

TextBox

TextBox bileşeni HTML input text etiketine denk ve veri girişi için kullanılır. TextBox bileşeninin ***text*** niteliği textbox nesnesinin barındırdığı veriyi tanımlar.

Örnek Form ve TextBox Bileşenlerinin kullanımı

View

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
    <title>Ornek2</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<faces>
    <@import taglib="phpf.ui.*" prefix="f"/>
    <f:form name="kayitformu" method="post">
        Uye adi : <f:textbox name="uyeadi"/>
        E-Posta : <f:textbox name="eposta"/>
        <f:button name="gonder" text="Gonder" onclick="actionevent"
forname="kayitformu"/>
    </f:form>
</faces>
</body>
</html>
```

Örnek Kontrolcü

```
<?php
import("phpf.controllers.facete");
class Ornek2 extends Facete {
    public function Ornek2() {
        parent::Facete();
        $this->render("ornek2.html");
    }
    protected function gonderClicked($olay) {
        echo "Gönderilen uye adi :".$this->uyeadi->text;
        echo "<br> Gönderilen E-Posta :".$this->eposta->text;
    }
}
?>
```

Gonerilen uye adi :huseyin
Gönderilen E-Posta :bora@huseyin.com
Uye adi : huseyin E-Posta : bora@huseyin.com Gonder

TextArea

TextArea bileşeni uzun metinler için veri girişi sağlar kullanımı HTML deki textarea gibidir. Barındırdığı veriye textbox nesnesinde olduğu gibi text niteliğinden erişilebilir.

Örnek TextBox örneğinde oluşturduğumuz form nesnesine adres bilgilerinin girişinde kullanmak üzere birde TextArea bileşeni ekleyelim.


```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
    <title>Ornek3</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<faces>
    <@import taglib="phpf.ui.*" prefix="f"/>
    <f:form name="kayitformu" method="post">
        Uye adi : <f:textbox name="uyeadi"/>
        E-Posta : <f:textbox name="eposta"/>
        <br>
        Adres : <f:textarea name="adres"></f:textarea>
        <f:button name="gonder" text="Gonder" onclick="actionevent"
forname="kayitformu"/>
    </f:form>
</faces>
</body>
</html>
```

Kontrolcü

```
<?php
import("phpf.controllers.facete");
class Ornek3 extends Facete {
    public function Ornek3() {
        parent::Facete();
        $this->render("ornek3.html");
    }
    protected function gonderClicked($olay) {
        echo "Gonderilen uye adi :".$this->uyeadi->text;
        echo "<br> Gönderilen E-Posta :".$this->eposta->text;
        echo "<br> Gönderilen Adres :".$this->adres->text;
    }
}
?>
```

Password

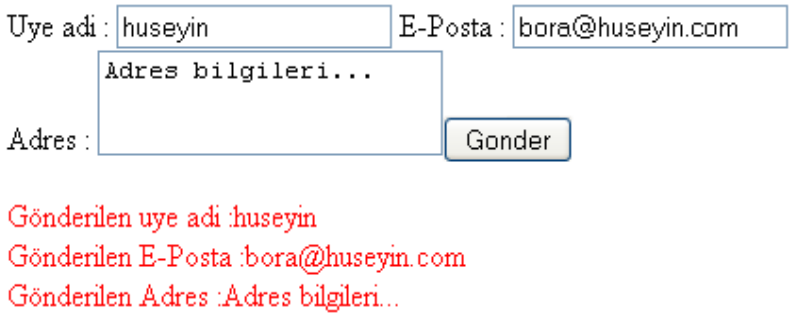
Pasword bileşeni TextBox bileşeni gibi çalışır ardaki tek fark Password girilen bilgilerin gorunmemesidir.

Hidden

Hidden bileşeni kullanıcıya görünmeyen gizli bilgilerin saklandığı bir bileşendir özellikle bilgilerin kullanıcaya gösterilmediği durağan ajax uygulamalarında fayda sağlar.

Label

Label etiket bileşeni bir yazı eklemek ya da AJAX kullanımında içerisini doldurmak üzere kullanılabilir.
Önceki örneklerdeki kayıt formunu Label ve AJAX kullanarak geliştirelim.



Görünüm

```
<html>
    <head>
    <title>Ornek4</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<faces>
    <@import taglib="phpf.ui.*" prefix="f"/>
    <f:form name="kayitformu" method="post">
        Uye adi : <f:textbox name="uyeadi"/>
        E-Posta : <f:textbox name="eposta"/>
        <br>
        Adres : <f:textarea name="adres"></f:textarea>
        <f:button name="gonder" text="Gonder" onclick="ajaxevent"
forname="kayitformu"/>
    </f:form>
    <font color="red">
        <f:label name="etiket">
        </f:label>
    </font>
</faces>
</body></html>
```

Kontrolcü

```
<?php
import("phpf.controllers.facete");
class Ornek4 extends Facete {
    public function Ornek4() {
        parent::Facete();
        $this->render("ornek4.html");
    }
    protected function gonderClicked($olay) {
        $veri= "Gonerilen uye adi :".$this->uyeadi->text;
        $veri.= "<br> Gönderilen E-Posta :".$this->eposta->text;
        $veri.= "<br> Gönderilen Adres :".$this->adres->text;
        $this->etiket->text= $veri;
        $this->AjaxResponse();
    }
}
?>
```

ComboBox

ComboBox bileşeni bir seçim kutusu oluşturur HTMLdeki select etiketine karşılık gelir bu bileşen ile view tarafında onchange niteliği ve kontrolcü tarafında Changed metodları kullanılabilir. Changed metodu değerlerin değişmesini yakalar.ComboBox nesnesinin ***getSelected()*** metodu seçili olan değeri barındırır. ComboBox bileşenin içini doldurmanın 3 yolu bulunmaktadır. Aşağıdaki örneklerde bu üç yönteme de değinilmektedir.

Birinci örneğimizde şehir seçimi için bir combobox ve seçilen şehri göstermek için bir Label bileşenimiz bulunmaktadır. Şehir ne zaman değişirse onchange olayı meydana gelir ve kontrolcü tarafındaki Changed metodu çalışır. Bu örnekte ComboBox bileşenin içini doldurmak için option etiketleri kullanılmaktadır.

Görünüm

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
    <title>Option</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<faces>
    <@import taglib="phpf.ui.*" prefix="f"/>
    Şehirler :
    <f:combobox id="sehirler" onchange="ajaxevent">
        <option>Ankara</option>
        <option>İstanbul</option>
        <option>İzmir</option>
        <option>Erzurum</option>
        <option>Van</option>
        <option>Diyarbakır</option>
        <option>Çankırı</option>
    </f:combobox>
    Seciminiz :
    <font color="blue">
        <f:label id="secim"></f:label>
    </font>
</faces>
</body>
</html>
```

Kontrolcü

```
<?php
import("phpf.controllers.facete");
class Ornek5 extends Facete {
    public function Ornek5() {
        parent::Facete();
        $this->render("ornek5.html");
    }
    protected function sehirlerChanged($olay) {
       $this->secim->setText($this->sehirler->getSelected());
       $this->AjaxResponse();
    }
}
```

?>

Yukarıdaki aynı örneği bu sefer ***bind*** niteliğini kullanarak uygulayacağız Bileşenlerin ***bind*** niteliği bir dizi ya da nesne kabul eder. Bind kullanımı için aşağıdaki örneği inceleyiniz.

Örnek Bind kullanımı.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
    <title>Option</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<faces>
    <@import taglib="phpf.ui.*" prefix="f"/>
    Şehirler :
    <f:combobox id="sehirler" onchange="ajaxevent"
                bind="array(Ankara,İstanbul,İzmir,Erzurum,Van,Diyarbakır,Çankırı)">
    </f:combobox>
    Seciminiz :
    <font color="blue">
        <f:label id="secim"></f:label>
    </font>
</faces>
</body>
</html>
```

Üçüncü örneğimizde seçime bağlı içeriği değişen bir combobox bileşeni oluşturuyoruz. Bu örnekte birinci combobox tan seçilen şehir bilgisine göre ikinci combobox'un içini bu şehirdeki ilçeler ile dolduracağız.bunun için kontrolcü tarafında combobox nesnesinin setModel metodunu kullanacağız.

Görünüm

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
    <title>Option</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<faces>
    <@import taglib="phpf.ui.*" prefix="f"/>
    Şehirler :
    <f:combobox id="iller" onchange="ajaxevent"
                bind="array(Ankara,İstanbul,İzmir,Erzurum,Van,Diyarbakır,Çankırı)">
    </f:combobox>
    İlçe :
    <f:combobox id="ilceler" onchange="ajaxevent"
                bind="array(Keçiören,Mamak,Çankaya,Yenimahalle,Sincan)">
```

```
    </f:combobox>
</faces>
</body>
</html>
```

Kontrolcü

```
<?php
import("phpf.controllers.facete");
class Ornek6 extends Facete {
    public function Ornek6() {
        parent::Facete();
        $this->render("ornek6.html");
    }
    protected function illerChanged($olay) {
        $tumilceler = array(
            "Ankara"=>array(Keçiören,Mamak,Çankaya,Yenimahalle,Sincan),
            "İstanbul"=>array( Adalar,Arnavutköy,Ataşehir,Bağcılar,Bakırköy),
            "İzmir"=>array( Adalar,Arnavutköy,Ataşehir,Bağcılar,Bakırköy),
            "Çankırı"=>array(Eldivan ,Ilgaz,Kurşunlu,Şabanözü),
            "Erzurum"=>array(Aşkale,Horasan,Karayazı,Pasinler)
        );

        $il = $this->iller->getSelected();
        if(array_key_exists($il,$tumilceler))
            $this->ilceler->setModel($tumilceler[$il]);
        else
            $this->ilceler->setModel(null);
        $this->AjaxResponse();
    }
}
?>
```

ComboBox nesnesinin ***setModel*** metodu ***Array*** veri tipi kabul eder ve içeriğini yeni değer ile değiştirir.

CheckBox

Checkbox bileşeni bir işaret kutusu oluşturur ve sunucu tarafına mantıksal ***true*** ya da ***false*** değerleri gönderir. Kontrolcü tarafında checkbox bileşeninin secili olup olmadığı isSelected metoduna bakarak anlaşılabilir.

Görünüm

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
    <title>Option</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<faces>
    <@import taglib="phpf.ui.*" prefix="f"/>
    İşaret kutusu <f:checkbox id="kutu" onclick="ajaxevent"/>
    Durum : <f:label id="durum"></f:label>
</faces>
</body>
</html>
```

Kontrolcü

```
<?php
import("phpf.controllers.facete");
class Ornek7 extends Facete {
    public function Ornek7() {
        parent::Facete();
        $this->render("ornek7.html");
    }
    protected function kutuClicked($olay) {
        $this->durum->text = $this->kutu->isSelected();
        $this->AjaxResponse();
}}
?>
```

Radio

Checkbox bileşeni gibi çalışır.

İmage

İmage bileşeni görüntü (resim) dosyalarının görüntülenmesi için kullanılır kullanımı HTML input image image gibidir. Görünüm dosyasında İmage bileşeninin src niteliğine gösterilmek istenen görüntü dosyasının yolu ve adi yazılır.

Örnek

```
<faces>
   <@import taglib="phpf.ui.*" prefix="f"/>
   <f:image name="resim"  src="resimler/resim1.png"/>
</faces>
```

Kontrolcü tarafında image bileşenin setSource(string dosya adı) metodu kullanılarak src niteliğindeki dosya adı değiştirilebilir.

Örnek AJAX resim galerisi
Bu örnekte resimler klasöründe resim1.jpg,resim2.jpg ,3,4,5 adlarında 5 adet resim bulunmakta ve görünüm dosyamızda da resimleri görüntülemek için bir image bileşeni, bir label ve resimleri değiştirmek içinde 5 adet button bileşeni bulunmaktadır.

Görümüm

```
<html>
    <head>
    <title>Resimler</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<faces>
    <@import prefix="f" taglib="phpf.ui.*"/>
    <f:image name="resim"  src="resimler/resim1.jpg" height="250"/>
    <br>
    Görüntülenen  : <f:label name="etiket"> Resim 1</f:label>
    <br>
    <f:button name="bir" text="1" onclick="ajaxevent"/>
    <f:button name="iki" text="2" onclick="ajaxevent"/>
    <f:button name="uc" text="3" onclick="ajaxevent"/>
    <f:button name="dort" text="4" onclick="ajaxevent"/>
    <f:button name="bes" text="5" onclick="ajaxevent"/>
</faces>
</body>
</html>
```

Kontrolcü actionPerformed metodu override edilmiştir.

```
<?php
import("phpf.controllers.facete");
class Resimler extends Facete {
    public function Resimler() {
        parent::Facete();
        $this->render("resimler.html");
    }
public function actionPerformed(ActionEvent $event) {
   $numara = $event->getComponent()->getText();
   $this->resim->setSource("resimler/resim$numara.jpg");
   $this->etiket->setText("Resim $numara");
   $this->AjaxResponse();
}
}?>
```

Link

Link bileşeni html deki a etiketinin karşılığıdır ve bu etiket gibi kullanılır yukarıdaki Galeri örneğinin link kullanarak yapılmasına ilişkin Görünüm dosyasının içeriği aşağıdadır.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
    <title>Option</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<faces>
    <@import prefix="f" taglib="phpf.ui.*"/>
    <f:image name="resim"  src="resimler/resim1.jpg" height="250"/>
    <br>
    Görüntülenen  : <f:label name="etiket" > Resim 1</f:label>
    <br>
    <f:link name="bir" text="1" onclick="ajaxevent"/>
    <f:link  name="iki" text="2" onclick="ajaxevent"/>
    <f:link name="uc" text="3" onclick="ajaxevent"/>
    <f:link name="dort" text="4" onclick="ajaxevent"/>
    <f:link name="bes" text="5" onclick="ajaxevent"/>
</faces>
</body>
</html>
```

Message

Label bileşenine benzer bir şekilde kullanılır genellikle doğrulama(validation) işlemlerinde kullandığı için Doğrulamalar bölümünde değinilecektir.

Gird

Veritabanı Model anlatılırken değinilecektir.

File Upload Bileşeni

File bileşeni tarayıcıda bir dosya upload kutusu oluşturur ve gönderilen dosyayı upload methodu ile sunucu üzerinde upload işlemini gerçekleştir. File etiketinin path niteliğine sunucuya trafnsfer edilen dosyanın kayıt edileceği dizin yazılmalıdır.

Örnek Dosya Upload

```
<html>
    <head>
    <title></title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<faces>
    <@import prefix="f" taglib="phpf.ui.*"/>
    <f:form id="form">
          <f:file id="dosya" path="resimler"/>
          <f:button id="b" text="Yükle" onclick="actionevent" forname="form"/>
    </f:form>
</faces>
</body>
</html>
```

```
<?php
import("phpf.controllers.facete");
class DosyaYukle extends Facete {
    public function DosyaYukle() {
        parent::Facete();
        $this->render("yukle.html");
    }

     function buttonClicked($evt){
         $this->resimler->upload();
     }
}
?>
```

Eğer file bileşeni ile belli dosya formatları gibi kıstaslarla yükleme işlemi yapmak isterseniz filevalidation kullanmanız gerekir bu konu ilerleyen sayfalarda doğrulamalar başlığında değinilecektir.

Bileşen Niteliklerine Controllerdan Erişim

Niteliklere erişirken nesne adı -> nitelik adı şeklindendir. Örneğin bir textbox birde button bileşenimiz olsun buttona tıklandığında textbox un rengini değiştirelim. faces view tanımlaması aşağıdaki gibidir.

```
<faces>
<@import taglib="phpf.ui.*" prefix="face"/>
<face:textbox name ="yazi"  text ="Merhaba"/>
<face:textbox name ="button"  text ="Rengi Değiştir"/>
</faces>
```

```
<?php
class Controller extends Facete {
    function Controller() {
        parent::FacesController();
        $this->render("view.html");
    }
    function buttonClicked($evt) {
        $this->yazi->setProperty("style", "background:blue");
    }
}

?>
```

Örneğimizde textbox bileşenin style niteliğine background:blue değerini aktardık ve rengini değiştirmiş olduk.

Bir niteliği değiştirmek istediğinizde

$this->kontrol_adi->nitelik=”deger”;

Ya da

$this-> kontrol_adi->setProperty(”nitelik”, “değer”);

İki kullanımda geçerlidir.
Şimdi örneğimizi biraz daha geliştirelim bir combobox bileşeninde renkler olsun ve seçilen rengi ajax kullanarak textbox nesnesine uygulayalım.

Örneğimiz için görüm dosyası aşağıdaki şekildedir.

```
<faces>
    <@import prefix="face" taglib="phpf.ui.*"/>
    <face:textbox name ="textbox"  text ="Merhaba"/>
    Rengi değiştir
    <face:combobox name="colors"
                   bind="array(Aqua,Blue,AliceBlue,Beige,Black,Brown
,Navy,Pink,Red,Orange,Yellow ,White,Green)"
                   onchange="ajaxevent">
    </face:combobox>
</faces>
```

Örneğimiz için kontrolcü dosyası aşağıdaki şekildedir

```
<?php
import("phpf.controllers.facete");
class Style extends Facete {
    function Style() {
        parent::Facete();
        $this->render("view.phpf");
    }
    protected function colorsChanged($evt) {
        $color = $this->colors->getSelected();
        $this->textbox->setProperty("style","background:$color");
        $this->AjaxResponse();
    }
}
?>
```

Bileşen mimarisi (kendi bileşenlerinizi geliştirin)

Bir faces bileşeni (component) oluşturmak için sınıfınızı Component sınıfından genişletirsiniz.

Basit bir bileşenin en az iki metodu bulunmalıdır. Kurucu metod ve startTag metodu.

Kurucu metodun iki parametresi bulunur birinci parametresi bir FacesController ikinci parametresi ise view dosyasında etiket olarak tanımlanan bileşenin nitelikler dizisidir.

function construct(FacesController &$controller,$args=null)

startTag metodu bileşen etiketinin başlangıç için çıktı üreten metodudur. Bir etiket açıldığında çalıştırılır.

endTag metodu bileşen etiketinin kapatıldığında çalıştırılan metodu.

Basit bir örnek

```
<?php
import("phpf.ui.component");
class Kalinyaz extends Component {
    function Kalinyaz (FacesController &$controller,$args=null) {
        parent::Component($controller,$args);
    }
    function startTag() {
        return '<strong>';
    }
    function endTag() {
        return '</strong>';
    }
}
?>
```

Örneğimizde Kalinyaz isminde bir bileşen tanımladık <f:Kalinyaz> </f:Kalinyaz> etiketleri arasındaki metni kalın yazı tipi ile tarayıcıda göstermek istiyoruz. Bileşenimizi kullanmak istediğimiz dizin içerisine kalinyaz.php adı ile kayıt ediyoruz.

Kalinyaz bileşenini view içerisinde kullanımı.

```
<faces>
    <@import prefix="my" taglib="dizinadi.kalinyaz"/>
    <my:kalinyaz name="kalin">
        Bu yazı kalın olmalı
    </my:kalinyaz>
    Bu yazı normal
</faces>
```

Bileşen içerisinde Niteliklere erişim.

Bileşenin attributes uyesi nitelikleri içeren bir dizidir nitelik değerlerine

$this->attributes[nitelikadi] şeklinde erişilebilir.

Yukarıdaki örneğimizi biraz daha geliştirip metin rengi ve metin boyunu uygulayan bir bileşen haline getirelim. Aşağıdaki örneği inceleyin

```
<?php
import("phpf.ui.component");
class Kalinyaz extends Component {
    function Kalinyaz (FacesController &$controller,$args=null) {
        parent::Component($controller,$args);
    }
    function startTag() {
        $html = '<strong>'.'<font color="'.$this->attributes[renk].
            '" size="'.$this->attributes[boy].'">' ;
        return $html;
    }
    function endTag() {
        return '</font></strong>';
    }
}
?>
```

Yukarıda bileşenimize boy ve renk adında iki nitelik olduğunu görülebilir.

Bileşenimizin view içerisinde kullanımı.

```
<faces>
    <@import prefix="my" taglib="dizinadi.kalinyaz"/>
    <my:kalinyaz name="kalin" boy="10" renk="red">
        Her hangi bir yazı
    </my:kalinyaz>
</faces>
```

Component sınıfının doAttributes() metodu tüm nitelikleri eş değer ismi ile yerleştirir

```
function startTag() {
    return '<strong>'.'<font'. $this->doAttributes() .'>' ;
}
```

Bileşenlere eklenen öğeler bileşeninizin items uye dizinsinde yer alır
Items kullanımı için aşağıdaki örneği inceleyin

```
<?php
import("phpf.ui.component");
class Strong extends Component {
    function Strong(FacesController &$controller,$args=null) {
        parent::Component($controller,$args);
    }
    function startTag() {
        $html = '<strong>'.'<font'. $this->doAttributes() .'>' ;
        foreach($this->items as $item)
            $html.= '<br/><a href="'.$item[url].'">'.$item[title].'</a>';
        return $html;
    }
    function endTag() {
        return '</font></strong>';
    }
}
?>
```

View içerisinde kullanımı

```
<faces>
    <@import prefix="my" taglib="mypath.strong"/>
    <my:strong name="test" size="4" face="Arial" color="red">
        <@item url="http://www.google.com" title="google"/>
        <@item url="http://www.webmahsulleri.com" title="WebMahsulleri"/>
        bazı yazilar
    </my:strong>
</faces>
```

Statik Sınıf Etiketleri

(Kendi Etiketleriniz)

Dokümanın başlarında Core sınıfı anlatılarken de değinildiği gibi Statik etiketler Kontrolcü için bir yeni bir nesne oluşturmaz render çıktısı için bir string bilgi üretirler. Statik sınıflar bileşen olmadıkları için Kontrolcü tarafından da erişilemezler. Statik etiketlerin kodlandığı sınıflarda Statik metotlardan oluşmaktadır. Bu etiketlere de bu sebepten statik ismi verilmiştir.

Statik etiket sınıflarının etiket için kullanacakları metot isimleri önemlidir.
Etiketin açıldığında çalışacak olan statik metodun adı **start** etiket ismi **Tag** şeklindedir ve bir aray dizi parametresi kabul etmelidir.

static function startEtiketTag($args)

Etiketin kapatıldığında çalışacak olan statik metodun adı **end** etiket ismi **Tag** şeklindedir.

static function endEtiketTag()

Örnek :
Aşağıdaki örneğin phpfaces/phpf dizininde deneme. php adında kayıtlı olması uygun olacaktır.

```
<?php
class Deneme {
    static function startKalinTag($args) {
        $html = '<strong>'.'<font color="'.$args[renk].
            '" size="'.$args[boy].'">' ;
        return $html;

    }
    static function endKalinTag() {
        return '</font></strong>';
    }
    }
```

Örnek Deneme sınıfının görümüm içerisinde kullanılması

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
    <title>Duragan Etiket</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<faces>
    <@import prefix="c" taglib="phpf.deneme" type="static"/>
    <c:kalin renk="blue" boy="10" >dememe</c:kalin>
</faces>
</body>
</html>
```

Static sınıflar çok sayıda metod barındırabilirler aşağıda Deneme iki yeni metod eklenmiştir.

```
<?php
class Deneme {
    static function startKalinTag($args) {
        $html = '<strong>'.'<font color="'.$args[renk].
            '" size="'.$args[boy].'">' ;
        return $html;

    }
    static function endKalinTag() {
        return '</font></strong>';
    }
        static function startRenkliTag($args) {
        $renkler = array(Blue,AliceBlue,Black,Brown ,Pink,Red,Orange,Yellow ,Green);
        $yazi = trim($args[yazi]);
        $n= strlen($yazi);
        $html="";
        for($i=0;$i<=$n;$i++) {
            $anahtar =array_rand($renkler);
            $size = rand(1,10);
            $html .= '<font  size="'.$size.'"
color="'.$renkler[$anahtar].'">'.htmlspecialchars($yazi[$i])."</font>" ;
        }
        $html;
       return $html;
    }

    static function endRenkliTag() {
        return '';
    }}?>
```

Görünüm içerisinde kullanım

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
    <title>Option</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<faces>
    <@import prefix="c" taglib="phpf.deneme" type="static"/>
    <c:kalin renk="blue" boy="10" >Kalın Yaz</c:kalin>
    <br>
    <c:renkli yazi="Rengarenk Yaz"/>
</faces>
</body>
</html>
```

Template engine

Template kullanmanızın temel amacı HTML Vs Etiketler ile diğer kodları birbirinden ayrılmasının sağlanmasıdır. PHP faces'ın görünüm anlayısında sunucu üzerinde çalışan etiketler bulunduğu için alışıla gelmiş template kullanımını çok fazla aramazsınız.
Daha öncede bahsedilgibi gibi FDL içerisindeki PHP Faces ifadeleri #{ aralağına} yazılıyordu verileri görünmümde kulanmanın diğerbir yoluda böyledir. Görünüm içerisinde public ve protected üyelere this anahtarı ile erişebilirsiniz. Sınıfınızın üyesi olmayan değişkenleri ya da verileri görünüme aktarırken Kontrolcünüzün append metodunu kullanabilirsiniz.
append metodunun ilk parametresi görünümdeki değişken ismi için bir string ikinci parametresi ise verinin kendisidir.

Örnekler

$this->append("isim","Hüseyin Bora");
$soyad = "Abacı"
$this->append("soyad",$soyad);
$this->append("sayi",1001);

Yukarıda da belirtiğim gibi public ve protected üyelere görünüm içerisinde erişebilirsiniz. $this anahtarından sonra "." Nokta kullana bilirsiniz. Aşağıdaki örneği inceleyin

Örnek kontrolcü

```
<?php
import("phpf.controllers.facete");
class Ornek2 extends Facete {
    public $uye1="Genel";
    protected $uye2="Muhafaza";
    public function Ornek2() {
        parent::Facete();
        $this->append("ad", "Hüseyin Bora");
        $this->append("soyad", "ABACI");
        $this->append("sayi", 1001);
        $this->append("dizi", array(20,2,123,4100,23,0,1));
```

```
        $this->render("sayac.html");
    }
}
?>
```

Örnek Görünüm şablonu

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
    <title>Şablon</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<faces>
    <@import prefix="x" taglib="phpf.core" type="static"/>
    İsim : #{$isim}
    <br>
    Soyad : #{$soyad}
    <br>
    Sayi : #{$sayi}
    <br>
    Public uye : #{$this.uye1}
    <br>
    Protected uye : #{$this.uye2}
    <br>
    Dizi
    <x:foreach var="$dizi" item="$oge">
        <p> Deger :  #{$oge}</p>
    </x:foreach>
</faces>
</body>
</html>
```

View state kullanımı

Viewstate Nedir? PHP sayfanızda postback'ler arasında kaybetmek istemediğiniz verileri saklamanıza yarayan bir php faces konteyneridir.

PostBack Nedir? Sayfanın sunucuya gidip yeniden gelmesidir.

View state kullanımı sayesinde gorunum dosyalarınızda kalıcılık sağlayabilirsiniz. View state kullanımın temel amacı sayfanız post olduğunda verilerin kayıp olmamasını sağlamaktır.

View state ASP.Net kullanıcılarının aşina oldukları bir veri depolama yapısıdır buna ek olarak Java JSF da benzer bir yapıya sahipdir. View state yapısı hantal bir yapı olarak bilinir ancak şuan için geliştirilmiş daha kullanışlı bir yöntem bulunmamaktadır. Bu yüzden ister PHP ya da ASP.Net İster JSF kullanın View state kullanırken büyük boyutlardaki verilerinizi saklamamanızı öneririm bu sayfnızın ağırlaşmasına sebebiyet verebilir.

PHP Faces viewstate tanımları @definition direktifi ile kontrolcuye bildirilir

<@definition viewstate=”true” stroge=”xhtml”/>

definition direktifinin birinci niteliğine viewstate=”true” demekle viewstate’i etkin hale getirmiş olursunuz.

stroge niteliğine verdiğimiz değerile veriler için oluşturulucak depolama alanını belirtimiş olursunuz bu alan ***xhtml*** yani istemci tarafındaki tarayıcı ya da ***session*** sunucu tarafındaki oturum dosyası olabilir.

Kalıcılığını sağlamak istediğmiz bileşenleri tanımlarken ***placeholder*** niteliğine ***true*** değerini verimelisiniz.

View state kullanımına örnek olarak sayfanızın bir köşesinde kullanıcı adının verilmesi gerektiğini düşünelim.
Bu işlem için her seferinde kullanıcı adını veritabanındaki tablodan cekmek yerine View state kullanabilirsiniz.

Örnek

```
<faces>
<@import prefix="phpf" taglib="phpf.ui.*"/>
<@definition viewstate="true" stroge="session"/>
<phpf:label name="uyeadi" placeholder="true"></phpf:label>
</faces>
```

Kontrolcu içerisinde bir sefer uyeadi nı guncellememiz kâfidir.

$this->uyeadi->setText("XXX Hosgeldin");

Doğrulamalar (Validation)

Doğrulama tanımlamaları view(görünün) içerisinde bileşen niteliklerine atama yapılarak sağlanır. Doğrulama sonucları event (olay) nesnesinin niteliğine göre AJAX ya da sayfa post edilerek tarayıcıya gönderilir.

Doğrulama işlemlerinde bileşenlerin 5 niteliği bulunur.

Bunlar

* validator
* rule
* messagefor
* message
* success

Validator: bu niteliğe static metodlar barındıran doğrulama sınıfının ismi verilmelidir. io dizininde filtler ve validator isimlerimlerinde hazır iki doğrulama sınıfı bulunmaktadır.

Rule : Validator niteliği ile bildirilen doğrulama sınıfının doğrulma işlemi için kullandığı metodun ismidir. (doğrulama metodu)

Message : Doğrulama işlemin sonucunun (false) yanlış olması durumunda verilecek olan uyarı mesajdır. Örneğin “Hatali email adresi girdiniz ! ”

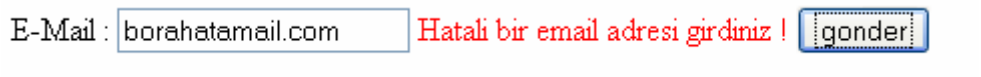
Success Doğrulama işlemin sonucunun (true) doğru olması durumunda verilecek olan mesajdır. . Örneğin “email adresiniz doğrulandı ! ” Eğer doğrulama mesajı verilmek istenmiyorsa bu nitelik boş bırakılabilir.

Messagefor : message ve success ile belirtilen mesajların hangi bileşende görüntüleneceğidir.

FacesController varsayılan durum da bir doğrulama işleminin sonucu yanlış ise olay metotlarını işletmez. Doğrulama işleminin sonucu yanlış olduğu halde örneğin bir Clicked metodunun işletilmesini istiyorsanız FacesController ın setValidCallBack(boolean); metoduna true değerini vermeniz yeterlidir.

Doğrulama işleminin sonucunun doğru ya da yanlış olduğunu bileşenin isValid metodunun döndürdüğü değere bakarak anlaşılabilir.

Örnek Kullanıcı tarafından girilen e-mail Adresinin doğrulanması.



View görünüm dosyası

```
<html>
    <head>
    <title>Validation</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<faces>
    <@import prefix="f" taglib="phpf.ui.*"/>
    <f:form method="post" id="form">
        E-Mail :<f:textbox id="email"
                           validator="validator"
                           rule="mail"
                           message=" Hatali bir email adresi girdiniz ! "
                           messagefor="msg"
                           />
        <font color="red">
            <f:message id="msg"/>
        </font>
        <f:button id="btn" text="gonder" onclick="ajaxevent" forname="form"/>
    </f:form>
</faces>
</body>
</html>
```

Kontrolcü

```
<?php
import("phpf.controllers.facete");
import("io.validator");
class Dogrula extends Facete {
    public function Dogrula() {
        parent::Facete();
        $this->render("dogrula.html");
    }
 protected function btnClicked($evt) {
     $this->msg->SetText("Email adresiniz Doğrulandı");
     $this->AjaxResponse();
 }}?>
```

Yukarıdaki kodlarda hatali bir email adresi girilmişse btnClicked metodu işletilmez mail adresinin doğrulanması durumunda bu metod çalışacaktır.

Doğrulama işlemi sonucu doğru ise msgbox isimli bileşende success ile belirtilen yanlış ise message ile belirtilen bilgi yer alır.

Filtler sınıfının metotları aşağıdaki gibidir.

- required
- mail
- url
- ip
- float
- int
- boolean
- regxp

Validator sınıfının metotları aşağıdaki gibidir.

- required
- equals
- minLength
- maxLength
- mail
- alpha
- alpha_numeric
- numeric
- integer
- betweenLength
- between
- boolean
- test

FileValidator Sınıfı

- Length
- Types
- Exists

Doğrulama Kullanımları(Validator sınıfı)

Doğrulama sınıflarunim kontroller içerisinde uygulamanıza dâhil edilmesi gerekmektedir.Validator sınıfı phpfaces/io/ dizini içerisindedir. Örnekler için kontrolcünüze aşağıdaki saturı eklemeyi unutmayın.

```
import("io.validator");
```

request post edilen bileşinin içeriğinin boş olup olmadığını kontrol eder
Örnek

```
isim :<f:textbox name="textbox"   validator=" validator" rule="required"  message="Bu
alan boş geçilemez" messagefor="msgbox"/>
<f:message  name="msgbox"/>
```

equals post edilen bileşenin içeriğinin test niteliği ile uyuşup uyuşmadığını kontrol eder
Örnek

```
isim :<f:textbox name="textbox"
                 validator="validator"
                 rule="equals"
                 test="200"
                 message="Hata girilen değer 200 değil "
                 messagefor="msgbox
                 "/>
<f:message  name="msgbox"/>
```

minLength post edilen bileşenin içeriğinin karakter uzunluğunun min niteliğinden büyük olup olmadığını kontrol eder
Örnek

```
isim :<f:textbox name="textbox"
                 validator="validator"
                 rule="minlength"
                 min="10"
                 message="En fazla 10 karakter "
                 messagefor="msgbox
                 "/>
<f:message  name="msgbox"/>
```

maxLength post edilen bileşenin içeriğinin karakter uzunluğunun max niteliğinden küçük olup olmadığını kontrol eder
Örnek

```
isim :<f:textbox name="textbox"
                 validator="validator"
                 rule="maxlength"
                 min=”5”
                 message="En az beş karakter girilmelisiniz !"
                 messagefor="msgbox
                 "/>
<f:message  name="msgbox"/>
```

Mail post edilen bileşenin içeriğinin bir mail adresi olup olmadığını kontrol eder
Örnek

```
isim :<f:textbox name="textbox"
                 validator="validator"
                 rule="mail"
                 message="Bu bir mail adresi değil !"
                 messagefor="msgbox”
                 "/>
<f:message  name="msgbox"/>
```

Alpha post edilen bileşenin içeriğinin bir sözel (string) olup olmadığını kontrol eder.
Örnek

```
isim :<f:textbox name="textbox"
                 validator="validator"
                 rule="alpha"
                 message="Girilen değer sözel değil !"
                 messagefor="msgbox”
                 "/>
<f:message  name="msgbox"/>
```

alpha_numeric post edilen bileşenin içeriğinin bir karakterlerden ve sayılardan oluşup oluşmadığını olmadığını kontrol eder.
Örnek

```
isim :<f:textbox name="textbox"
                 validator="validator"
                 rule="alpha_numeric"
                 message="Girilen değer sayı içermeli !"
                 messagefor="msgbox”
                 "/>
<f:message  name="msgbox"/>
```

Numeric post edilen bileşenin içeriğinin bir sayısal değer olup olmadığını kontrol eder.
Örnek

```
isim :<f:textbox name="textbox"
                 validator="validator"
                 rule="numeric"
                 message="Uyarı girdiğiniz değer bir sayı değil !"
                 messagefor="msgbox”
                 "/>
<f:message  name="msgbox"/>
```

integer post edilen bileşenin içeriğinin bir tam sayı olup olmadığını kontrol eder.
Örnek

```
isim :<f:textbox name="textbox"
                 validator="validator"
                 rule="integer"
                 message="Uyarı girdiğiniz değer bir tam sayı değil !"
                 messagefor="msgbox”
                 "/>
<f:message  name="msgbox"/>
```

betweenLength post edilen bileşenin içeriğinin uzunluğunun min ve max değerleri arasında olup olmadığını kontrol eder.

Örnek

```
isim :<f:textbox name="textbox"
                 validator="validator"
                 rule="betweenLength"
                 min="5"
                 max="20"
                 message="Beş karakterden az Yirmi karakterden uzun veri girmeyiniz! "
                 messagefor="msgbox”
                 "/>
<f:message  name="msgbox"/>
```

Çoklu Doğrulamalar(Multi Validation)

Çoklu doğrulamalar bir bileşene birden çok doğrulama yapmak istediğiniz zaman kullanışlıdır çoklu doğrulamalarda nitelik değerleri “,” virgül ile bir birlerinden ayrılmalıdır. Örneğin required ve mail doğrulamalarının birlikte planlanması.

```
<f:textbox name="email"
           validator="validator"
           rule="mail,required"
           message="* Hatali mail adresi,* Bu alan boş geçilemez"
           success="Adres doğrulandı,"
           messagefor="msg1,msg2"/>
<f:message id="msg1"/>
<f:message id="msg2"/>
```

Bu örnekte önce mail doğrulaması daha sonra required doğrulaması yapılır. her iki doğrulama sonucu yanlış olursa her iki mesajda işletilir.
mail doğrulamasının mesajı “msg1″ required doğrulamasının mesajı “msg2″ bileşenlerine yazdırılır.
e-mail adresinin doğrulanması durumunda “Adres doğrulandı” mesajı msg1 bileşeninde yazdırılır.

FileValidator ve File Bileşeni ile Upload

FileValidator file bişeleni ile birlikte kullanılır örneğin belli boyuttun üzerindeki dosyaların yüklenmemesini ya da belli dosya formatlarının yüklenmesi gerekebilir. FileValidator sınıfının 3 metodu bulunur length (kabul edilen en fazla dosya boyutu)tyeps (Kabul edilen dosya formatları) exists(aynı dosya adında başkabir dosanın blunması durumu)

Örnek Boyutu 50000 den küçük olan GIF ve JPG formatlarını sunucu ya yüklemek

```
<html>
    <head>
    <title></title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<faces>
    <@import prefix="f" taglib="phpf.ui.*"/>
    <f:form id="form">
```

```
          <f:file id="file" path="resimler/"    length="50000"
validator="filevalidator" rule="types,length" types="image/jpeg,image/gif"
message="Desteklenmeyen Dosya Formatı,50000 byte tan büyük dosya"
messagefor="hata,hata"/>
         <f:message id="hata"/>
         <f:button id="b" text="Yükle" onclick="actionevent" forname="form"/>
    </f:form>
</faces>
</body>
</html>
```

```
<?php
import("phpf.controllers.facete");
class DosyaYukle extends Facete {
    public function DosyaYukle() {
        parent::Facete();
        $this->render("yukle.html");
    }

     function buttonClicked($evt){
         $this->resimler->upload();
     }
}
?>
```

Doğrulama Sınıfları (Kendi Doğrulayıcılarınız)

Doğrulayıcı(Validator) sınıfları durağan(static) metotları kullanan sınıflardır. işlerinizi kolaylaştırması ve yazılımın bütünleşik parçalar halinde kolektif bir yapıda çalışmasına olanak sağlar.
Bir doğrulama metodunun tanımlaması aşağıdaki gibidir

static function required(Component $c,Component $m,$message,$success)

Birinci parametre “Component $c” doğrulama işleminin yapıldığı bileşen
İkinci parametre “Component $m” mesajların yazdırılacağı bileşen
Üçüncü parametre uyarı mesajı
Dördüncü parametre doğrulama mesajı
Örneğin yeni üye kaydı için bir üye kayıt formumuzun olduğunu ve aynı ismi taşıyabilecek sadece bir üye olduğunu düşünelim.

Bunun için Uyekontrol adında bir sınıf oluşturup varmi ismini taşıyan bir doğrulama metodu ekliyorum

```
<?php
class uyekontrol {
    static function varmi(Component $c,Component $m,$message,$success) {
```

```
        $sorgu =  EntityManager::getInstance()->createQuery("select  u from uyeler u
where uye.adi =:uyead");
        $sorgu->bindParam("uyeadi", $c->getText);
        $sorgu->execute();
        $sonuc =$sorgu->getSingle();
        if($sonuc!=null) {
            $m->setText($message);
            return false;
        }
        $m->setText($success);
        return true;
    }
}
?>
```

Şimdi oluşturduğumuz doğrulama sınıfını view (görünüm) içerisinde kullanalım

```
<faces>
    <@import prefix="f" taglib="phpf.ui.*"/>
    <f:form method="post" name="uyeform">
        Uye adi :<f:textbox
            name="uyeadi"
            validator="uyekontrol"
            rule="varmi"
            success="Uye adi doğrulandı "
            message=" Daha önce bu isimde bir üye kayıt olmuş farklı bir isim deneyin!"
            messagefor="msgbox"
            />
        <f:message id="msgbox"/>
        <f:button name="kaydet" text="Gönder" onclick="actionevent" forname="uyeform"/>
    </f:form>
</faces>
```

Validador sınıfımızı controller içerisinde import ediyoruz

```
<?php
import("phpf.controllers.facete");
import("dbf.persistence");
import("dogrulayicilar.uyekontrol",true);//uygulama dizini
class Controller extends Facete {
    function Controller() {
        parent::Facete();
        $this->render("uyekayit.phpf");
    }
    function kaydetClicked($evt) {
        $uye = new uye();
        $uye->adi = $this->uyeadi->text;
        $uye->save();
    }
}
?>
```

Doğrulamalarda Olayları İşletmek

Doğrulama işleminin sonucu false döndürdüğünde varsayılan durumda olay metodları işletilmez ancek isterseniz işletilmesini isteyebilirsiniz bu durumda render metodundan önce kontrolcünüzün ***setValidCallBack(boolean);*** metodunu true parametresi vererek çalıştırmanız gerekmektedir.

Doğrulama işleminin durumunu ilgili bileşenin ***isValid()*** metoduna bakarak öğrenebilirsiniz. Bu metot doğruluk sağlanmışsa true sağlanmamışsa false değerlerinden birini verir.

Örnek görünüm

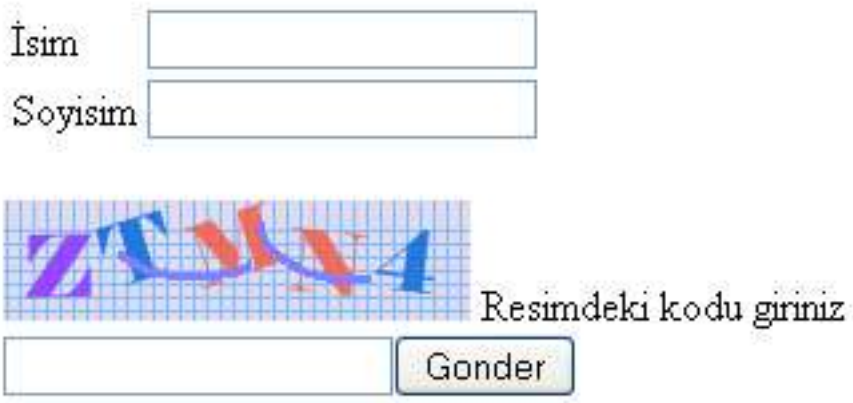
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
    <title>Dogrula</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<faces>
    <@import prefix="f" taglib="phpf.ui.*"/>
    <f:form id="form" method="post">
        <f:textbox name="email"
                   validator="validator"
                   rule="required"/>
        <f:message id="msg1"/>
        <f:button id ="btn" onclick="ajaxevent" forname="form" text="Gonder"/>
    </f:form>
</faces>
</body>
</html>
```

Örnek Kontrolcü

```
<?php
import("phpf.controllers.facete");
import("io.validator");
class Dogrula extends Facete {
    public function Dogrula() {
        parent::Facete();
        $this->setValidCallBack(true);
        $this->render("dogrula.html");
    }
    protected function btnClicked($e) {
        if(!$this->email->isValid())
            $this->msg1->text= "Bu alan boş geçilemez";
        $this->AjaxResponse();
    }
}
?>
```

Captcha Bileşeni Doğrulama Resimleri

Captcha bileşeni bir kareter içeren bir resim oluşturu ve kullanıcının doğrulamasını ister. Bu bileşen validation doğrulamalar ile beraber kullanılmalıdır.



Örnek Capthca Güvenlik kodu

```
<html>
    <head>
    <title></title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<faces>
    <@import prefix="f" taglib="phpf.ui.*"/>
     <f:form method="post" id="form">
        <table>
            <tr>
                <td>İsim</td>
                <td> <f:textbox id="isim"/></td>
            </tr>
            <tr>
                <td>Soyisim</td>
                <td> <f:textbox id="soyisim"/></td>
            </tr>
        </table>
        <br>
        <f:captcha id="captcha"/>
        <f:message id="msg" text="Resimdeki kodu giriniz"/>
        </br>
        <f:textbox id="code" validator="validator" rule="equals"
                   test="#{$this.captcha.code}"
                   message="Resimdeki kodu yanlış girdiniz!"
                   messagefor="msg"
                   />
        <f:button id="b" text="Gonder" onclick="actionevent" forname="form"/>
    </f:form>
</faces>
</body>
</html>
```

Load Metodu Dinamik Üye Nesneler

Facete ve FacesController'un load metodu birinci parametresi ile verilen dosya ismiyle aynı ismi taşıyan ve kurucusu parametre içermeyen sınıftan bir örnek oluşturur ve Kontrolcüye bir public üye olarak ekler. İmport fonksiyonuna benzer bir kullanımı vardır.

Örnek

```
<?php
import("phpf.controllers.facete");
import("io.validator");
class Ornek2 extends Facete {
    public function Ornek2() {
        parent::Facete();
        $this->load("server.session");
        $this->session->get("userid");
    }
}
?>
```

Oluşturulacak yeni nesnenin ismini belirtebilirsiniz. Load metodunun ikinci parametresi oluşturulacak uyenin ismidir.

Örnek

```php
<?php
import("phpf.controllers.facete");
import("io.validator");
class Ornek2 extends Facete {
    public function Ornek2() {
        parent::Facete();
        $this->load("server.session","oturum");
        $this->oturum->get("userid");
    }
}
?>
```

Load metodunun üçüncü parametresi true ya da false değerlerinden birini kabul eder. Bu parametre true verilirse ilgili dosyaya uygulama dizini içerisinde bakılır.

URI Sınıfı

URI sınıfı path dizin şeklindeki url satırlarından gelen verileri elde etmek ve sayfayı yönlendirmek için kullanılır.
Sayfa adında bir kontrolcümüzün olduğunu düşünürsek kontrolcüye erişmek için gerekli olan URL satırı
Aşağıdakine benzer bir şekilde olacaktır.

http://localhost/phpfaces/sayfa

URL satırında kontrolcü adının ardından " / " karekterleri arasındaki veriler parametre sayılır ve "/" sayısı kadar bir dizi oluşturulur.
Örneğin ***http://localhost/phpfaces/sayfa/1 sondaki 1 parametresi 0'ıncı indistedir.***

Örnek

```php
<?php
import("phpf.controllers.facete");
class Sayfa extends Facete {
    public function Sayfa() {
        parent::Facete();
        $this->load("io.uri");
     echo " Parametre 0 = ".   $this->uri->get(0);
    }
}
?>
```

Örneğin ***http://localhost/phpfaces/sayfa/kitaplar/beyaz_gemi/1125145*** URLsatırını ele alalım

Kitaplar 0. indiste
Beyazgemi 1.indiste
1125145 2.indiste

Örnek

```php
<?php
import("phpf.controllers.facete");
class Sayfa extends Facete {
    public function Sayfa() {
```

```
        parent::Facete();
        $this->load("io.uri");
        echo " Kategori = ".   $this->uri->get(0);
        echo " Kitap adı  = ".   $this->uri->get(1);
        echo " ISBN  = ".   $this->uri->get(2);

    }
}
?>
```

URI sınıfının redirect metodu parametre olarak verilen URL adresini tarayıcıya yönlendirmesi için söyler.

Örnek

```
<?php
import("phpf.controllers.facete");
class Sayfa extends Facete {
    public function Sayfa() {
        parent::Facete();
        $this->load("io.uri");
        $this->uri->redirect("http://www.google.com.tr");
    }
}
?>
```

Render işlemi ve Header nesnesi

FacesRenderer sınıfı faces dosyasını ayrıştırken <head> etiketi ile karşılaştığında otomatik olarak bir Header nesnesi oluştur render işleminin sonucunda da bu header nesnesini döndürür. Header nesnesi sayfanın <head> </head> aralığına ekleme yapmak için kullanılır.
Header sınıfının metotları şöyledir

- addMeta(string name, string content) görüm'e bir meta etiketi ekler
- addScript(string file) görüm'e file ile belirtilen dosya adı ile script etiketi ekler
- addLink(string file) görüm'e file ile belirtilen dosya adı ile link etiketi ekler rel="stylesheet" type="text/css" niteliklerini kullanır
- setTitle(string title) sayfa başlığını değiştirir
- addText(string text) text parametresini <head> </head> etiketleri arasına ekler.

Örnek

```
<?php
require_once ("config.php");
import("phpf.controllers.facete");
class Head extends Facete {
    public function Head() {
        parent::Facete();
        $this->render("view.html");
    }
    function dugmeClicked($e) {
        $head =FacesRenderer::getHeader();
        $head->setTitle("Butona Tıklayınca Başlğı değiştirdim");
    }
    public function prerender() {
        $head= FacesRenderer::getHeader();
        $head->addMeta("description", "PHP Faces MVC Framework");
        $head->addMeta("keywords", "PHP,ORM,MVC,AJAX");
        $head->addScript("scritp.js");
        $head->addLink("style.css");
    }
```

```
}
Dispatcher::dispatchclass("head");
?>
```

Görünüm

```
<html>
    <head>
    <title></title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<faces>
    <@import taglib="phpf.ui.button" prefix="f"/>
    <f:button name="dugme" text="Tıkla1" onclick="actionevent"/>
</faces>
</body>
</html>
```

Renderer Directives İşleyici Direktifleri

Önceki yazılarda da belirtiğim gibi FDL içerisinde renderer direktiflerinin tamamı @ işareti ile başlarlar. Bunlar

* @definition
* @import
* @pattern
* @ui
* @htmlpattern
* @html
* @set
* @get
* @item
* @include
* @face

@definiton

Bu direktifin 3 niteliği vardır

- viewstate : true ya da false değerlerinden birini alır,görünüm içerisinde bileşen verileri için bir depolama alanı oluşturur bu sayde sayfa postback olduğunda veriler kalıcı olur. ASP.Net deki viewstate e benzer bir yapısı vardır. dikkatli kullanılması gerekelidir. çok büyük veriler işlem güçünü yavaşlatabilir. varsayılan değeri false dir

- stroge : viewstate deki verinin nerede tutulacağıdır viewstate ile birlikte kullanılır xhtml ya da session değerlerinden birini alır. bu niteliğe xhtml değeri verilse depo alanı istemci tarafındaki tarayıcıdır. session olması durumunda sunucu tarafındaki oturum dosyalarıdır.

- eventvalidation : istemci tarfından gelen eventlara bir zaman aşımı ve bir anahtar ile doğrulunu kontrol eder. varsayılan değeri false dir

Örnek

```
<faces>
    <@definition viewstate="true" stroge="xhtml"/>
    <@definition eventvalidation="true"/>
</faces>
```

@import

3 niteliği vardır.

- prefix : etiketler ile kullanılacak isim uzayı
- taglib: import edilecek olan dizin ve ya php dosyası
- type: bileşen kütüphanesinin tipi

Bu direktif renderer a import methodunu işletmesini, prefix ile ilişkili isim ile ilişkilendirmesini söyler.

type niteliği sadece static sınıflarda kullanılır ve PHP Faces da şuan tek static kütüphane core.php dir

Örnek

```
<faces>
    <@import prefix="c" taglib="phpf.core" type="static"/>
    <c:out value="Merhaba Faces"/>
</faces>
```

Başka bir örnek

```
<faces>
    <@import prefix="c" taglib="phpf.core" type="static"/>
    <@import prefix="f" taglib="phpf.ui.*"/>
    <c:if test="1<2">
        <c:out value =" bir ikiden kucuktur"/>
    </c:if>
    <f:button name="button" text="tıkla"/>
</faces>
```

@set

@set direktifi faces yorumlayıcısına bir değişken üretmesini ve bir değer atamasını söyler.iki niteliği vardır.

Birinci niteliği üretilecek olan değişkenin adı

İkinci niteliği bu değişkenin nerede saklanacağıdır. page ya da session değerlerinden birini alır.

Kullanımı aşağıdaki gibidir.

<@set sayi="2008" scope="page"/>

Yukarıdaki satır ile sayi adında bir değişken oluşturulur, bu değişkene 2008 değeri atanır ve o anki gecerli betik içerisinde saklanır.

<@set isim="bora" scope="session"/>

yukarıdaki satır ile isim adında bir değişken oluşturulur, bu değişkene bora değeri atanır ve sunucudaki oturum dosyasında saklanır.

@get

get direktifi faces yorumlayıcısına ber değişken üretmesini ve bu değişkenin değerini bir başka değişkenden atamasını söyler. Üç niteliği vardır. Birinci nitelik değişken adının bildirildiği var niteliğidir. İkinci nitelik hangi değişkenden atama yapılacanın belirtildiği select niteliğidir. Üçüncü nitelik ise değişkenin konumunun belirtildiği scope niteliğidir session ya da page değerlerinden birini alır.

<@get var="istesayi" select="sayi" scope="page"/>
<@get var="isteisim" select="isim" scope="session"/>

@include

Bu direktif renderer a bir dosyayı view e eklemesini söyler php dosyasıda olabilir

Kullanımı aşağıdaki gibidir.

<@include file="functions.php"/>
<@include file="helper.php"/>
<@include file="view.php"/>

@face

Bu direktif renderer a bir faces dosyasını işlemesini view e eklemesini söyler

Kullanımı aşağıdaki gibidir.

<@face file="patterns.phpf"/>
<@face file="footer.phpf"/>
<@face file="banner.html"/>

@item

@item direktifi bir bileşen etiketine yeni öğeler ekler. @item başka bir etiketin içerisinde kullanılır. ve kendine özgü niteliklri barındırır.

Kullanım örnekleri

```
<sql:sql name="sql" var="results" query="select * from table where id=:id">
    <@item id="{$id}" type="integer"/>
</sql:sql>
```

```
<com:grid name="girdx" height="100" width="50" bind="$this.birler" border="1">
    <@item  input="text" title="id" key="id" />
    <@item  input="button" key="name" title="name"/>
    <@item  input="link" title="delete"  url="delete/$id/$name/$id" type="path"
separator="_"/>
    <@item  input="link" title="edit"  url="delete.php?id=$id&amp;name=$name"/>
</com:grid>
```

Patternler

pattern kavramı programlamada işaretleme dilleri açısından bir yeniliktir. ve bu yöntem ilk defa PHP faces ile tanıtılmaktadır.

patternler özetle bir veri tanımlaması biçimde bir şablon oluşturur ve bu tanımın uygulanmasını sağlar
Daha anlaşılır bir şekilde. Farklı bir kopyalama ve yapıştırma işlemi görmektedir. patternler sayesinde programcı birirlerine benzeyen etiket (tag) bloklarını tekrarından kurtulur.

PHP faces içerisinde 2 çeşit pattern vardır bunlar:
@pattern
@htmlpattern

@Pattern

@pattern PHP Faces bileşenleri içindir ve uygulanırken @ui direktif etiketi ile kullanılır.

@htmlpattern içerisindeki metni biraz farklı bir biçimde kopyalar içerisine HTML, XML vb yerleştirilebilir. @htmlpattern , @html direktif etiketi ile kullanılır.
@pattern

@pattern direktif etiketinin kendisine haz üç niteli vardır. bu nitelikten sonra gelecek nitelikler genişletilen faces bileşenine has niteliklerdir.

- name: niteliği tanımlanan pattern (şablonun) ismidir.
- prefix: niteliği genişletilcek bileşenin hangi isim uzayında bulunduğunu bildirir
- extends: niteliği genişletilecek faces bileşeninin adıdır.

Örnek

```
<faces>
    <@import prefix="f" taglib="phpf.ui.button"/>
    <@pattern name="mybutton"
    prefix="f"
    extends="button"
    value="her hangi bir şey"/>
</faces>
```

Yukarıdaki örnekte button bileşeninden mybutton isminde yeni bir pattern tanımlanmış.
@ui

- direktif etikeki @pattern ile tanımlanan bir şablonu uygular. İki niteliği vardır.
- name : yeni oluşturulacak olan bileşenin adıdır.

pattern: kullanılacak olan pattern tanımlamasının adıdır.

Örnek patternnin uygulanması

```
<faces>
    <@ui name="mybutton1" pattern="mybutton"/>
    <@ui name="mybutton2" pattern="mybutton"/>
    <@ui name="mybutton3" pattern="mybutton"  value="farklı bir yazı"/>
    <@ui name="mybutton4" pattern="mybutton" style="color:navy;font-weight:bold;" >
</faces>
```

yukarıdaki 1. ve 2. satırlarda mybutton patterninden 2 örnek üretilmiş üçüncü satırta da mybutton patterninden yeni bir örnek üretilip value niteliğine “farklı bir yazı atanmış 4. satırda da mybutton pattern ninden yeni bir örnek üretilip style niteliğne color:navy;font-weight:bold; değeri atanmıştır.

sonucta html çıktısı aşağıdaki gibidir.

```
<input type="button" name = "mybutton1"  id="mybutton1" value="her hangi bir şey"/>
<input type="button" name = "mybutton2"  id="mybutton2" value="her hangi bir şey"/>
<input type="button" name = "mybutton3"  id="mybutton3" value="farklı bir yazı"/>
<input type="button" name = "mybutton4"  id="mybutton4" value="her hangi bir şey"
style="color:navy;font-weight:bold;"/>
```

@htmlpattern

yukarıda belirtiğim gibi htmlpattern içine gelen şablon metini kopyalar ve @html direktifi ile şablon uygulanır.

name niteliği oluşturulan patternin ismidir. @html direkttif etiketinin pattern niteliğine burada belirtilen isim yazılır.

Örnek @htmlpattern tanımlaması

```
<faces>
    <@htmlpattern name="html">
    <div style="color:red;font-weight:bold;" > Senin Yazın  </div>
    </@htmlpattern>
</faces>
```

Örnek @html ile htmlpatternlerin uygulanması

```
<faces>
    <@html pattern="html"/>
</faces>
```

Yukarıdaki örneğin çıktısı şu şekilde olur.

```
<div style="color:red;font-weight:bold;" > Senin Yazın  </div>
```

paterrnleri istediğiniz sayıda kullana bilirsiniz

Örnek @html ile htmlpatternlerin uygulanması

```
<faces>
    <@html pattern="html"/>
    <@html pattern="html"/>
    <@html pattern="html"/>
</faces>
```

Yukarıdaki örneğin çıktısı şu şekilde olur.

```
    <div style="color:red;font-weight:bold;" > Senin Yazın  </div>
    <div style="color:red;font-weight:bold;" > Senin Yazın  </div>
    <div style="color:red;font-weight:bold;" > Senin Yazın  </div>
```

html patternler ile birlikte php faces ifadeleride kullanılabilir
php faces ifadeleri #{ ve } aralığına yazılır.

Örnek başka bir htmlpattern tanımlaması

```
<faces>
    <@htmlpattern name="html2">
    <div style="color:red;font-weight:bold;">test htmlpattern  #{$i}</div>
    </@htmlpattern>
</faces>
```

Yukarıdaki pattern nin uygulanması

```
<faces>
    <c:for var="$i" begin="1" to="5" step="1">
        <@html pattern="html2"/>
    </c:for>
</faces>
```

Örneğin çıktısı

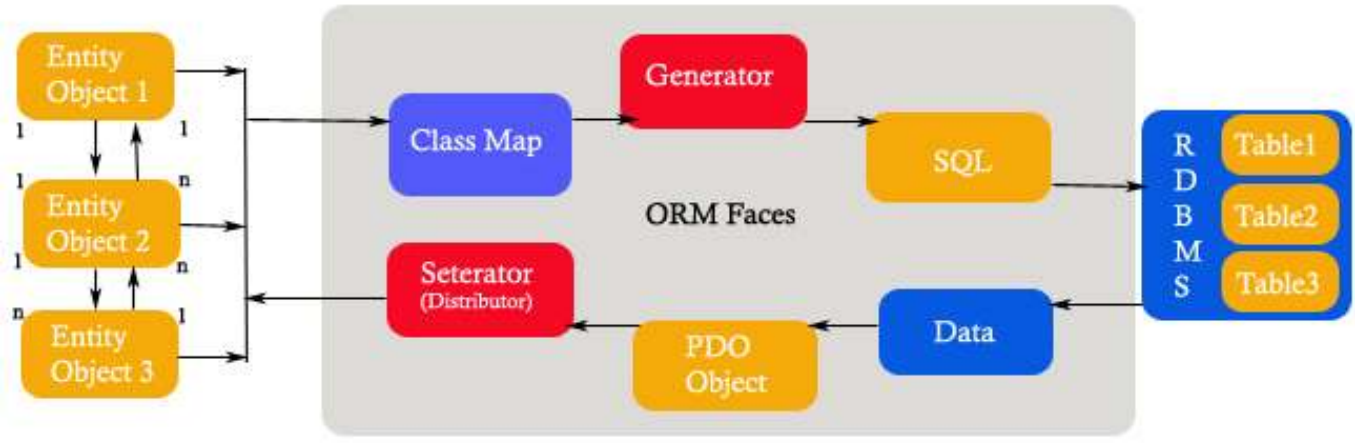
```
    <div style="color:red;font-weight:bold;">test htmlpattern  1</div>
    <div style="color:red;font-weight:bold;">test htmlpattern  2</div>
    <div style="color:red;font-weight:bold;">test htmlpattern  3</div>
    <div style="color:red;font-weight:bold;">test htmlpattern  4</div>
    <div style="color:red;font-weight:bold;">test htmlpattern  5</div>
```

ORM Faces

PHP ORM Faces

ORM nedir? İlişkisel Nesne Eşleştirme (Object Relation Mapping) basit bir anlatımla veritabanındaki tablolara karşılık gelen sınıflar ve bu sınıfların özelliklerine göre SQL cümleleri üretebilen bir yazılım mimarisidir diyebiliriz.

Tablolarımıza karşılık gelen bu sınıflara genel anlamda **Varlık**(Entity) sınıfları adını veriyoruz. Nesne yönelimli programlamada her şey birer nesne olarak düşünülmeli O halde veri tabanındaki işlemlerimizde nesneler olarak düşünmeliyiz. İşte burada ORM bize veritabanındaki tablolarımızı nesneleştirmemizi ve **CRUD**(create, read, update, delete) işlemlerimizi bir nevi basitleştirmemizi
sağlamakta.



Aşağıda bazı ufak örnekler var

```
$user = new User("Username","password");
```

```
$user->save();
$user = EntityManager::getInstance()->find("User",12);
$user->delete();
```

Veri tabanı yapılandırması
applications/config.php dosyasını bir metin editorü ile düzenlemek üzere açın burada PHP Faces ORM veritabanı erişim yapılandırması bulunur.Aşadaki satırları kendi sisteminze göre düzenleyin.

```
define("DB_CONNECTION_STRING","mysql:host=sunucuadi;dbname=veritabanı");
define("DB_USER","kullanıcıadı");
define("DB_PASS","parola");
```

Faces Entity (Varlıklar)

Entity Varlık sınıfları Entity sınıfından genişletilir. Genellikle import işlemlerini controllerda yaparsınız. Aşağıdaki satırların controller da bulunması iyidir.

```
import("dbf.persistence");
import("models",true);
```

import fonksiyonun ikinci parametresinin true verilmesi durumda bu fonksiyon uygulamaya dahil edilecek dosyalara application/uygulamaadi/ içerisinede bakar.

Tekrar konumza yani varlık sınıflarına dönelim. varlık sınıflarını veri tabanındaki tablolarınız ile ilişkilendirirken haritalarken annotationları kullanırsınız.

Annotationlar

@Table(name = "blog") tablo adı tanımlanır class sözcüğün üzerine yazılır
Name : parametresine tablo adı yazılır

@Column(name = "name" ,type = "string") alan tanımlaması yapılır

Name :niteliğine alan adı yazılır. ilişkili private üyenin üzerine yazılmaldır.
üye private olmalıdır bu önemlidir çünki Faces ORM private üyelere tanır.

@Id birincil anahtarı tanımlar

ilişkisel annotationlar

@OneToOne

@OneToMany

üç paremetreleri bulunur

mappedBy = "Sınıf adı"

pk = ” primary key ”

fk = ” foreign key”

Olayı daha iyi kavrayabilmeniz için örneği inceleyin.

CREATE TABLE `blog` (
`id` int(11) NOT NULL auto_increment,
`name` varchar(255) default NULL,
`content` text,
PRIMARY KEY (`id`)
)

Yukarıdaki gibi bir tablomuz olsun.

```
<?php
/**
 *  @Table(name = "blog")
 */
class Blog extends Entity {
/**
 @Id
 @Column(name = "id" ,type = "integer")
 */

    private $id;
    /**
     @Column(name = "name" ,type = "string")
     */

    private $name;
    /**
     @Column(name = "content" ,type = "text")
     */

    private $content;

    public function set($name, $value) {
        $this->$name= $value;
    }

    public function get($name) {

        return $this->$name;
    }
}

?>
```

Entity sınıfında hangi alanların güncellendiğini tespit etmek için __set ve __get metotları aşırı yüklenmiştir. bu yüzden set ve get metotlarını kullanın

Dilerseniz her üye için get ve set metotlarınıda uygulaya bilirsiniz Faces ORM her iki yaklaşımıda benimser.

Varlık siniflarınızı application/uygulamaadi/models/ dizinine yerleştirin application/uygulamaadi/models/blog.php gibi

Gelelim Controller içerisinden varlık nesnelerimize nasıl erişeceğimize.

```
<?php
import("phpf.controllers.facescontroller");
import("dbf.persistence");
import("models.*",true);//application/models dizini
class BlogController extends FacesController {
    public function  BlogController () {
        parent::FacesController();
        $blog = new Blog();
        $blog->name="Hello World";
        $blog->content="Some text";
        EntityManager::getInstance()->save($blog);
    //EntityManager::getInstance()->delete($blog);
    }
}
?>
```

Yukarıdaki satırlarda yeni bir Blog nesnesi oluşturulur değerler atanır ve son olarak blog nesnesi veri tabanına insert edilir.eğer blog nesnesi new ile kendi oluşturduğumuz bir nesne olmasaydı bu onun update edileceği anlamına gelmekteydi EntityManager::save(Entity) metodu varlık nesnesinin veritabındanmı yoksa yeni bir nesnemi olduğunu algılar ve bu duruma göre ya insert ya da update sözcükleri oluşturup veritabanı sunucusuna gönderir.

Varlık sınıflarını her seferinde elinizle yazmak zorunda değilsiniz. Bunun için geliştirdiğim ufak bir java uygulaması var. http://code.google.com/p/php-faces/downloads/list adresinden generator.zip dosyasını indirin. uygulamayı kullanabilmeniz için sisteminizde java vm nin kurulu olması kafidir. generator programı sizin için varlık sınıflarını oluşturacaktır.

Faces ORM Varlık Sınıfları Arasında İlişkiler

ilişkisel annotationlar

@OneToOne

@OneToMany

@ManyToOne

@ManyToMany

üç paremetreleri bulunur

mappedBy = “Sınıf adı”

pk = ” primary key ”

fk = ” foreign key”

Aşağıdaki örneği inceleyin

CREATE TABLE `blog` (

`id` int(11) NOT NULL auto_increment,

`name` varchar(255) default NULL,

`content` text,

PRIMARY KEY (`id`)

)

CREATE TABLE comment` (

`comentid` INT NOT NULL AUTO_INCREMENT PRIMARY KEY ,

`content` TEXT NOT NULL ,

`blogid` INT NOT NULL

)

Yukarıdaki yapı şeklinde iki tablomuz olsun
blog ile comment arasında bire e çok
comment ile blog arasında bire e bir ilişki bulunur.

Buna göre varlık sınıflarımızı aşağıdaki gibi tanımlamız gerekir.

Blog sınıfı

```
<?php
/**
 *  @Table(name = "blog")
 */
class Blog extends Entity {
/**
 @Id
 @Column(name = "id" ,type = "integer")
 */

    private $id;
    /**
     @Column(name = "name" ,type = "string")
     */

    private $name;
    /**
     @Column(name = "content" ,type = "text")
     */
```

```
    private $content;
    /**
     @OneToMany(mappedBy = "Comment",pk="id",fk="blogid")
     */

    private $comments;//ArrayObject

    function Blog() {
        parent::Entity();
        $this->comments= new ArrayObject();
    }

    public function set($name, $value) {
        $this->$name= $value;
    }

    public function get($name) {

        return $this->$name;
    }
}?>
```

Comment sınıfı

```
<?php
/**
 *  @Table(name = "comment")
 */
class Comment extends Entity {
/**
 @Id
 @Column(name = "commentid" ,type = "integer")
 */

    private $id;
    /**
     @Column(name = "content" ,type = "text")
     */

    private $content;
    /**
     * @Column(name = "blogid",type ="INT")
     * @OneToOne(mappedBy = "blog", pk= "id", fk= "blogid")
     */

    private $blogid;

    public function set($name, $value) {
        $this->$name= $value;
    }

    public function get($name) {

        return $this->$name;
    }
}
?>
```

Controller sınıfımız

```
<?php
import("phpf.controllers.facescontroller");
import("dbf.persistence");
import("models.*",true);
```

```
class Blogtest extends FacesController {

    public function  Blogtest () {
        parent::FacesController();
        $blog = new Blog();
        $blog->name="Hello World";
        $blog->content="Some text";
        $comment = new Comment();
        $comment->content= "New Comment";
        $blog->comments->append($comment);
        EntityManager::getInstance()->save($blog);
    }
}

?>
```

Eğer varlık sınıflarınızı Faces Generator ile oluşturmak isterseniz ilişkilerinizi önceden SQL sorgularınızda bildirmeniz yerinde olacaktır. Faces Generator ilişkileri tanır ve buna göre sınıflar oluşturur.

Faces ORM Entity Manager (Varlık Yöneticisi)

Entity Manager (Varlık Yöneticisi) isminden de anlaşılacağı üzeri varlık sınıflarınız üzerinde temel işlemleri gerçekleştirmenizi sağlar. varlık yöneticisi ister controller sınıfınızda isterseniz entity sınıflarınız içersinde kullanın.
Metotlar

bir varlığı veri tabanına kaydetmek duruma göre insert ya da update sözcükleri üretilir

EntityManager::save(Entity $e)

Örnek

$blog = new Blog();
$blog->name="Hello world";
$blog->content="xxxx";
EntityManager::getInstance()->save($blog);

bir varlığı veri tabanınıda aramak için EntityManager'ın find metotdu kullanılır. Bu metodun ilk parametresi aranacak varlık sınıfı ikinci parametersi bu varlığa karşılık gelen birincil anahtar primary key dir.

EntityManager::find(Entity $e,int Id)

Örnek

$em = EntityManager::getInstance();
$blog = $em->find("Blog",2);
echo $blog->name;
echo $blog->content;

bir varlığı veri tabanınıdan silmek için EntityManager'ın delete metotdu kullanılır.

Entity EntityManager::delete(Entity $e)

Örnek

```
$em = EntityManager::getInstance();
$blog = $em->find("Blog",2);
$em->delete($blog);
```

FQL sorguları işletmek için EntityManager'ın createQuery metotdu kullanılır. Bu methot string bir parametre alır ve bir Query nesnesi dödürür.

Query nesnesi PDOStatement nesnesinden genişletilmiştir.

FQL le ilerki yazılarda değiniceğim.

Query EntityManager::createQuery(String FQL)

Örnek

```
$em= EntityManager::getInstance();
$query = $em->createQuery("select b from Blog b where b.id =:id");
$id=11;
$query->bindParam("id", $id);
$query->execute();
print_r($query->getResultList())
```

SQL sorguları işletmek için EntityManager'ın nativeQuery metotdu kullanılır. Bu methot string bir parametre alır ve bir Query nesnesi dödürür.

Query nesnesi PDOStatement nesnesinden genişletilmiştir.

Query EntityManager::nativeQuery(String SQL)

Örnek

```
$em= EntityManager::getInstance();
$query = $em->createQuery("select* from blog");
$query->execute();
```

Query nesneleri PDOStatement'i genişletiği için incelemenizi öneririm.

PDO Transaciton ları destekler

```
beginTransaction();
commit();
rollBack();
```

Örnek

Varlık sınıfı içinde EntityManager kullanımı

```
<?php
/**
 *  @Table(name ="comment")
 */
class Comment extends Entity {
/**
 @Id
 @Column(name = "id" ,type = "integer")
 */

    private $id;
    /**
     @Column(name = "content" ,type = "text")
     */

    private $content;

    public function set($name, $value) {
        $this->$name= $value;
    }

    public function get($name) {

        return $this->$name;
    }
    function save() {
        $em = Manager::getInstance();
        $em->beginTransaction();

        try {
            $em->save($this);
            $em->commit();
        } catch (Exception $e) {
            $em->rollBack();
        }
    }
    function delete() {
        $em = Manager::getInstance();
        $em->beginTransaction();

        try {
            $em->delete($this);
            $em->commit();
        } catch (Exception $e) {
            $em->rollBack();
        }
}}

?>
```

Örnek Yukarıdaki Varlık sınıfının kontrolcü içerisinde kullanılması

```
<?php
import("phpf.controllers.facescontroller");
import("dbf.persistence");
import("models.*",true);
class Commenttest extends FacesController {

    public function  Commenttest () {
        parent::FacesController();
        $blog = new Comment();
        $blog->content="bazı yazılar";
        $blog->save();
```

```
        echo "kayıt tamamdır ";
}}
?>
```

Faces ORM FQL (Faces Query Language)

FQL varlık nesneleri üzerinde sorgulama yapmanıza olanak tanıyan SQL e benzer bir sorgulama dilidir. FQL Sorguları PHP Faces Framework tarfından doğal SQL sözcüklerine dönüştürülür.

SELECT İfadesi

SELECT takmaisim FROM sıfıfadı takmaisim WHERE having vb..

Örnekler
SELECT b from Blog b

SELECT c from Categories c

SELECT b from Blog b WHERE bi.id = 1

SELECT b from Blog b WHERE bi.id = 1 and b.name='www'

SELECT b from Blog b limit 0,5

SELECT c from Categories c GROUP BY c.id HAVING AVG(c.id) > 1

entity manager ile beraber FQL kullanımı

```
$query= $this->em->createQuery("SELECT b from Blog b WHERE b.comment.id >1");
$query->execute();
$blogs = $query->getResultList();
```

FQL ile birlikte SQL fonksiyonları kullanımı

SELECT COUNT(b.id) FROM Blog b

SELECT MAX(b.id) FROM Blog b

entity manager ile beraber kullanımı

```
$query= $this->em->createQuery("SELECT count(b.id) from bir b");
$query->execute();
echo "count =" .$query->getSingle()."<br>";
```

İç içe geçmiş alt sorgular

SELECT b FROM Blog b WHERE b.coment.id =(SELECT MAX(c.id) FROM Comment c)
SELECT o FROM Object o WHERE o.id = (SELECT AVG(i.id) FROM Object i) ORDER BY o.name

Grid Bileşeni

Grid bileşeni kendisine bind niteliği ile belirtilen Sınıf diziden bir HTML tablosu oluşturur. Oluşturulan tabloda hangi sutunların bulunacağı @item direktifi ile bildirilir. @item direktifinin key niteliğine Varlık sınıfının uye ismi title niteliğine ise sutunun başlığı yazılır. @item direktifi ile tabloya sınıf harici vilgi eklemek için input niteliği kullanılır.

<@item input="link" title="Düzenle" url="testet.php?edit=$no"/>

Örneğin yukarıdaki @item direktifi ile düzenle başlıklı bir bağlantı oluşturulur ve bağlantı adresi url niteliği ile belirtilir.

Örnek Aşağıdaki gibi bir Varlık sınfımız olsun

```
<?php
/**
*  @Table(name = "urun")
*/
class Urun extends Entity
 {
/**
* @Id
* @Column(name = "no")
*/
 private   $no;

/**
* @Column(name = "ad")
*/
 private   $ad;

/**
* @Column(name = "fiyat")
*/

private   $fiyat;

function get($name) {
return $this->$name;
}

function set($name,$value) {
```

```
$this->$name = $value;
}
}
?>
```

Urunlerin listesini oluşturmak için aşağıdaki gibi bir kontrolcümüz olsun $list değişkenin içerisine veritabanından çektiğimiz bilgileri aktarıyoruz.

```
<?php
import("phpf.controllers.facete");
import("dbf.persistence");
import("entityes.urun",true);
class UrunGrid extends Facete {
    protected $list;
    public function UrunGrid() {
        parent::Facete();
        $em = EntityManager::getInstance();
        $query= $em->createQuery("Select u from urun u");
        $query->execute();
        $this->list = $q->getResultList();
        $this->render("grid.html");
    }
}
?>
```

Görünüm Dosyamız grid etiketinin bind niteliğine bind="$this.list" eklememiz önemli

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title></title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

</head>
  <body>
<faces>

      <@import prefix="f" taglib="phpf.ui.*"/>
      <f:grid name="grid" height="30%" width="75%" bind="$this.list" border="1">
         <@item   key="no"  title="no" />
         <@item   key="ad" title="Urun adı"/>
         <@item   key="fiyat" title="fiyat"/>
         <@item  input="link" title="Edit"  url="testet.php?edit=$no"/>
     </f:grid>
  </faces>
  </body>
</html>
```

no	Urun adı	fiyat	Edit
2	deneme14433	5555111	Edit
3	denememe11	1111	Edit
4	222	3434	Edit

SQL Etiketi

SQL etiketi query niteliğine paremetre olarak verilen SQL sorgusu işletir ve id niteliğine verilen isim ile bir sonuc kümesi getirir.

Örnek

```
<html>
    <head>
    <title></title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<faces>
    <@import prefix="c" taglib="phpf.core" type="static"/>
    <@import prefix="sql" taglib="phpf.db.sql"/>
    <sql:sql id="results" query="select * from urun"/>
    <table border="1" align="center">
        <thead>
        <th>Urun NO</th>
        <th>Urun Adı</th>
        <th>Fiyat</th>
        <th>Resim</th>
        </thead>
        <c:foreach var="#{$this.list}" item="$urun">
            <tr>
                <td>#{$urun.no}</td>
                <td>#{$urun.ad}</td>
                <td>#{$urun.fiyat}</td>
                <td><img src="#{$urun.aciklama}"/></td>
            </tr>
        </c:foreach>
    </table>
</faces>
</body>
</html>
```

FQL Etiketi

FQL etiketi query niteliğine paremetre olarak verilen FQL sorgusu işletir ve id niteliğine verilen isim ile bir sonuc kümesi getirir.

Örnek

```
<html>
    <head>
    <title></title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<faces>
    <@import prefix="f" taglib="phpf.ui.*"/>
    <@import prefix="fql" taglib="phpf.db.fql"/>
    <fql:fql id="results" query="select u from urun u"/>
    <f:grid name="grid" height="30%" width="75%" bind="$results" border="1">
        <@item   key="no"  title="no" />
        <@item   key="ad" title="Urun adı"/>
        <@item   key="fiyat" title="fiyat"/>
    </f:grid>
</faces>
```

```
</body>
</html>
```

SQL ve FQL etiketlerine @item etiketinin value niteliği ile parametre aktarılabilir.
Örnek fiyat aralığı

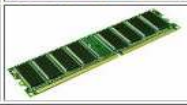
```
<html>
    <head>
    <title></title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<faces>
    <@import prefix="f" taglib="phpf.ui.*"/>
    <@import prefix="fql" taglib="phpf.db.fql"/>
    <fql:fql id="results" query="select u from urun u WHERE u.fiyat >? AND u.fiyat <?">
    <@item value ="100"/>
    <@item value= "500"/>
    </sql:fql>

    <f:grid name="grid" height="30%" width="75%" bind="$results" border="1">
        <@item   key="no"  title="no" />
        <@item   key="ad" title="Urun adı"/>
        <@item   key="fiyat" title="fiyat"/>
    </f:grid>
    <f:classpager id="pager" request="page" limit="3" count="10" />
</faces>
</body>
</html>
```

Pager Etiketi

Pager Etiketi sayfalama yapmak amacıyla kullanılır.

İki farklı kullanım vardır birinci kullanım şeklinde entity niteliğine varlık sınıfının adı. Limit niteliğine gösterilecek kayıt sayısı yazılır eğer FQL etiketinde olduğu gibi bir sonuc kümesi elde edilmek isteniyora resource niteliğine geri alınacak sounuc kümesinin adı yazılmalıdır.

Urun no	Urun adı	fiyat	Resim	Detay
2	CPU	100		Detay
3	Ana kart	80		Detay
4	RAM	50		Detay

Eğer php faces framework ü URI modunda kullanıyorsanız pager nesnesinin uri biçimde sayfalama oluşturması için pager etiketinin type niteliğine path değerini vermeniz gerekir.

Örnek Pager ve Grid Kullanımı

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    </head>
    <body>
    <faces>
        <@import prefix="f" taglib="phpf.ui.*"/>
        <@import prefix="db" taglib="phpf.db.*"/>
        <f:grid id="grid" width="100%" bind="$liste" border="1">
            <@item   key="no"  title="Urun no" />
            <@item   key="ad" title="Urun adı"/>
            <@item   key="fiyat" title="fiyat"/>
            <@item  input="img"  key="resim" title="Resim"/>
            <@item  input="link" title="Detay"  url="testet.php?edit=$no"/>
        </f:grid>
        <db:pager id="pager"  limit="3" entity="urun" resource="liste"
previous="Önceki" next="Sonraki"/>
    </faces>
</body>
</html>
```

Pager etiketinin query niteliği FQL secim işleminden sonraki FQL cümlelerini kabul eder
Örnek Fiyat aralığı

```
<db:pager id="pager"  type="path" limit="3" entity="urun" resource="liste"
                      query="WHERE urun.fiyat >=50 AND urun.fiyat<=100"/>
```

Pager bileşeninin diğer kullanımı ise veritabanı sorgularını Kontrollcü içerisinde yapıp pager bileleşenine count niteliğne toplan kayıt sayısını ve limit niteliğinede sayfada gösterilecek kayıt sayısı verilmelidir.

Örnek
Görünüm

```
<html>
    <head>
    <title></title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<faces>
    <@import prefix="c" taglib="phpf.core" type="static"/>
    <@import prefix="db" taglib="phpf.db.*"/>

    <table border="1" align="center">
        <thead>
        <th>Urun NO</th>
        <th>Urun Adı</th>
        <th>Fiyat</th>
        <th>Resim</th>
        </thead>
```

```
        <c:foreach var="#{$this.list}" item="$urun">
            <tr>
                <td>#{$urun.no}</td>
                <td>#{$urun.ad}</td>
                <td>#{$urun.fiyat}</td>
                <td><img src="#{$urun.aciklama}"/></td>
            </tr>
        </c:foreach>
    </table>
    <db:pager id="pager" limit="3" count="10"/>
</faces>
</body>
</html>
```

Controller

```
import("phpf.controllers.facete");
import("dbf.persistence");
import("entityes.*",true);
import("phpf.db.pager");
class Urunler extends Facete {
    protected $list;
    public function Urunler() {
        parent::Facete();
        $this->render("gorunum/test.html");
    }
    public function prerender() {
        $em = EntityManager::getInstance();
        $query= $em->createQuery("Select count(u.no) from urun u");
        $this->pager->setCount($query->execute()->getSingle());
        $query= $em->createQuery("Select u from urun u GROUP BY u.no ".$this->pager-
>getLimitQuery());
        $this->list =$query->execute()->getResultList();
    }
}
?>
```

Pager bileşeninin QueryString den aldığı p değişkeni istenirse değiştirile bilir bunun için bilşen tanımı view dosyasında yapılırken request niteliğne işlenecek değişken adı yazılır.

```
<db:pager id="pager" limit="3" count="10" request="sayfa"/>
```