FREE eBook

LEARNING primefaces

Free unaffiliated eBook created from
Stack Overflow contributors.

#primefaces

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: primefaces

It is an unofficial and free primefaces ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official primefaces.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with primefaces

Remarks

PrimeFaces is an open source JSF framework. Its main features:

- 100+ components.
- Built-in Ajax based on standard JSF Ajax APIs.
- Push support via Atmosphere Framework.
- Mobile UI kit to create mobile web applications.
- 35+ built-in themes.
- Premium themes and layouts.

Versions

Version	Release Date
0.8.1	2009-02-23
0.8.2	2009-03-26
0.8.3	2009-04-23
0.9.0	2009-06-15
0.9.1	2009-08-04
0.9.2	2009-09-07
0.9.3	2009-10-05
1.0.0 and 2.0.0	2010-02-15
1.0.1 and 2.0.1	2010-04-19
1.0.2 and 2.0.2	2010-05-31
1.1 and 2.1	2010-07-26
2.2	2011-02-07
3.0	2012-01-04
3.1	2012-02-06
3.2	2012-03-12
3.3	2012-05-29

Version	Release Date
3.4	2012-09-03
3.5	2013-02-04
4.0	2013-10-03
5.0	2014-05-05
5.1	2014-10-06
5.2	2015-04-08
5.3	2015-10-19
6.0	2016-06-07

Examples

Installing PrimeFaces

PrimeFaces can be used in all web applications based on Java Server Faces (version 2.x) which are run on Servlet Containers (e.g. Wildlfy or Tomcat or GlassFish).

There are several ways you can add PrimeFaces to your application.

Manually

Download the `primefaces-{version}.jar` and add it to you classpath.

Maven

```
<dependency>
    <groupId>org.primefaces</groupId>
    <artifactId>primefaces</artifactId>
    <version>{version}</version>
</dependency>
```

For older versions (3.5 and below) you additionally have to add the PrimeFaces repository:

```
<repository>
    <id>prime-repo</id>
    <name>PrimeFaces Maven Repository</name>
    <url>http://repository.primefaces.org</url>
    <layout>default</layout>
</repository>
```

Gradle

```
repositories {
    mavenCentral()
        maven {
            url "http://repository.primefaces.org"
        }
}

dependencies {
    compile "org.primefaces:primefaces:{version}"
}
```

NetBeans

PrimeFaces is bundled with the Java EE bundle of NetBeans. When you create a new "Java Web -> Web Application" you can select JavaServer Faces as framework. Then you configure JSF to use PrimeFaces components. It will copy the library to your project.

If you have created a Maven web application, you can select project properties and select JavaServer Faces as framework and then select PrimeFaces as mentioned above. Your `pom.xml` will be modified to include the PrimeFaces dependency.

Hello world

After adding PrimeFaces to your JSF project, you can start using it in your pages using the namespace:

```
xmlns:p="http://primefaces.org/ui"
```

or, for PrimeFaces Mobile:

```
xmlns:p="http://primefaces.org/mobile"
```

This example should render a spinner:

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:p="http://primefaces.org/ui">
    <h:head>
    </h:head>
    <h:body>
        <p:spinner />
    </h:body>
</html>
```

Read Getting started with primefaces online: https://riptutorial.com/primefaces/topic/1027/getting-started-with-primefaces

Chapter 2: Hello Primefaces

Examples

Hello Primefaces

This is a simple application with primefaces, it is a login page:

1-Configuration of **web.xml**:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
    <context-param>
        <param-name>javax.faces.PROJECT_STAGE</param-name>
        <param-value>Development</param-value>
    </context-param>
    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>*.xhtml</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>index.xhtml</welcome-file>
    </welcome-file-list>
</web-app>
```

2-Create **ManagedBean**:

```
import javax.faces.application.FacesMessage;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;
import javax.faces.context.FacesContext;

@ManagedBean
@RequestScoped
public class LoginBean {

    private String username;
    private String password;

    public LoginBean() {
    }
```

```
    public void login() {
        //Simple login
        if (!this.username.equals("") && this.username.equals(password)) {
            FacesContext.getCurrentInstance().addMessage(null, new FacesMessage(
                    FacesMessage.SEVERITY_INFO, "Success", "Login with success"));
        } else {
            FacesContext.getCurrentInstance().addMessage(null, new FacesMessage(
                    FacesMessage.SEVERITY_ERROR, "Error", "Username or password not
correct"));
        }
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }
}
```

3-Create ***.xhtml page**:

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:p="http://primefaces.org/ui">
    <h:head>
        <title>Hello Primefaces</title>
    </h:head>
    <h:body>
        <h:form>
            <p:messages id="my_message" showDetail="true" autoUpdate="true" closable="true" />
            <h:panelGrid columns="2">
                <p:outputLabel value="UserName :" for="username"/>
                <p:inputText id="username" value="#{loginBean.username}" required="true"
                             requiredMessage="Username is required"/>
                <p:outputLabel value="Password :" for="password"/>
                <p:password id="password" value="#{loginBean.password}" required="true"
                            requiredMessage="Password is required"/>
                <span/>
                <p:commandButton value="Login" actionListener="#{loginBean.login}"/>
            </h:panelGrid>
        </h:form>
    </h:body>
</html>
```

4-Deploy your application.

5-open your browser and type: `http://localhost:8080/HelloPrimefaces/`

Read Hello Primefaces online: https://riptutorial.com/primefaces/topic/3309/hello-primefaces

Chapter 3: How to use Primefaces showcase

Examples

The example of Primefaces panelGrid component in its Showcase.

The showcase of Primefaces components you can find here and documentation is here

Frontend needs to be saved as a XHTML file. This file can contain JSF, JSTL, JSP, HTML, CSS, jQuery, javaScript and its framework and more front-end technologies. Please, do not mix JSF and JSP technologies together. It is not a good approach.

Note you have to define namespaces such as c, f, h, p, pe and so on in the beginning .

```
<h:form id="form">
    <p:dataGrid var="car" value="#{dataGridView.cars}" columns="3" layout="grid"
        rows="12" paginator="true" id="cars"
        paginatorTemplate="{CurrentPageReport}  {FirstPageLink} {PreviousPageLink} {PageLinks}
{NextPageLink} {LastPageLink} {RowsPerPageDropdown}"
        rowsPerPageTemplate="6,12,16">

        <f:facet name="header">
            Cars for Sale
        </f:facet>

        <p:panel header="#{car.id}" style="text-align:center">
            <h:panelGrid columns="1" style="width:100%">
                <p:graphicImage name="demo/images/car/#{car.brand}.gif"/>

                <h:outputText value="#{car.year}" />

                <p:commandLink update=":form:carDetail" oncomplete="PF('carDialog').show()"
title="View Detail">
                    <h:outputText styleClass="ui-icon ui-icon-search" style="margin:0 auto;"
/>
                    <f:setPropertyActionListener value="#{car}"
target="#{dataGridView.selectedCar}" />
                </p:commandLink>
            </h:panelGrid>
        </p:panel>

    </p:dataGrid>

    <p:dialog header="Car Info" widgetVar="carDialog" modal="true" showEffect="fade"
hideEffect="fade" resizable="false">
        <p:outputPanel id="carDetail" style="text-align:center;">
            <p:panelGrid  columns="2" rendered="#{not empty dataGridView.selectedCar}"
columnClasses="label,value">
                <f:facet name="header">
                    <p:graphicImage name="demo/images/car/#{dataGridView.selectedCar.brand}-
big.gif"/>
                </f:facet>

                <h:outputText value="Id:" />
                <h:outputText value="#{dataGridView.selectedCar.id}" />
```

```
                <h:outputText value="Year" />
                <h:outputText value="#{dataGridView.selectedCar.year}" />

                <h:outputText value="Color:" />
                <h:outputText value="#{dataGridView.selectedCar.color}"
style="color:#{dataGridView.selectedCar.color}"/>

                <h:outputText value="Price" />
                <h:outputText value="$#{dataGridView.selectedCar.price}" />
            </p:panelGrid>
        </p:outputPanel>
    </p:dialog>
</h:form>
```

Back-end needs to be saved as a JSF files. This file contains mostly Java, but there can be javaScript calls to Front-end. The annotation of class might be replaced with Spring configuration. **The View class** that it is used for serving data to Frontend EL (expression language).

```
package org.primefaces.showcase.view.data;

    import java.io.Serializable;
    import java.util.List;
    import javax.annotation.PostConstruct;
    import javax.faces.bean.ManagedBean;
    import javax.faces.bean.ManagedProperty;
    import javax.faces.bean.ViewScoped;
    import org.primefaces.showcase.domain.Car;
    import org.primefaces.showcase.service.CarService;

    @ManagedBean
    @ViewScoped
    public class DataGridView implements Serializable {

        private List<Car> cars;

        private Car selectedCar;

        @ManagedProperty("#{carService}")
        private CarService service;

        @PostConstruct
        public void init() {
            cars = service.createCars(48);
        }

        public List<Car> getCars() {
            return cars;
        }

        public void setService(CarService service) {
            this.service = service;
        }

        public Car getSelectedCar() {
            return selectedCar;
        }

        public void setSelectedCar(Car selectedCar) {
```

```
            this.selectedCar = selectedCar;
        }
    }
```

The service class serves data from DB but it is used in most of examples in PF Showcase to initialize and create data. Note that the most useful examples of code in the PF showcase are Frontend.

```
package org.primefaces.showcase.service;

import java.util.ArrayList;
import java.util.List;
import java.util.UUID;
import javax.faces.bean.ApplicationScoped;
import javax.faces.bean.ManagedBean;
import org.primefaces.showcase.domain.Car;

@ManagedBean(name = "carService")
@ApplicationScoped
public class CarService {

    private final static String[] colors;

    private final static String[] brands;

    static {
        colors = new String[10];
        colors[0] = "Black";
        colors[1] = "White";
        colors[2] = "Green";
        colors[3] = "Red";
        colors[4] = "Blue";
        colors[5] = "Orange";
        colors[6] = "Silver";
        colors[7] = "Yellow";
        colors[8] = "Brown";
        colors[9] = "Maroon";

        brands = new String[10];
        brands[0] = "BMW";
        brands[1] = "Mercedes";
        brands[2] = "Volvo";
        brands[3] = "Audi";
        brands[4] = "Renault";
        brands[5] = "Fiat";
        brands[6] = "Volkswagen";
        brands[7] = "Honda";
        brands[8] = "Jaguar";
        brands[9] = "Ford";
    }

    public List<Car> createCars(int size) {
        List<Car> list = new ArrayList<Car>();
        for(int i = 0 ; i < size ; i++) {
            list.add(new Car(getRandomId(), getRandomBrand(), getRandomYear(),
getRandomColor(), getRandomPrice(), getRandomSoldState()));
        }

        return list;
```

```
    }

    private String getRandomId() {
        return UUID.randomUUID().toString().substring(0, 8);
    }

    private int getRandomYear() {
        return (int) (Math.random() * 50 + 1960);
    }

    private String getRandomColor() {
        return colors[(int) (Math.random() * 10)];
    }

    private String getRandomBrand() {
        return brands[(int) (Math.random() * 10)];
    }

    public int getRandomPrice() {
        return (int) (Math.random() * 100000);
    }

    public boolean getRandomSoldState() {
        return (Math.random() > 0.5) ? true: false;
    }

    public List<String> getColors() {
        return Arrays.asList(colors);
    }

    public List<String> getBrands() {
        return Arrays.asList(brands);
    }
}
```

NOTE: Please, improve this documentation if you have enough relevant experience.

Read How to use Primefaces showcase online: https://riptutorial.com/primefaces/topic/6623/how-to-use-primefaces-showcase

Chapter 4: widgetVar

Introduction

widgetVar is the name of the client side variables which contains all the javascript PF widgets on the page. There is a great intro/tutorial to using the widgetVar component written by Hatem Alimam called Intro To PrimeFaces widgetVar

Examples

Basic usage of widgetVar

```
<h:form>
    <p:dialog widgetVar="myDialog"></p:dialog>
    <p:commandButton onclick="PF('myDialog').show();" />
</h:form>
```

Datatable

`datatable.js` in GitHub Reporitory

Function	Details
`bindPaginator: function()`	Binds the change event listener and renders the paginator
`loadLiveRows: function()`	Loads rows on-the-fly when scrolling live
`paginate: function(newState)`	Ajax pagination
`fetchNextPage: function(newState)`	Loads next page asynchronously to keep it at viewstate and Updates viewstate
`sort: function(columnHeader, order, multi)`	Ajax sort
`filter: function()`	Ajax filter
`onRowClick: function(event, rowElement, silent)`	
`onRowDblclick: function(event, row)`	
`highlightRow: function(row)`	Highlights row as selected
`unhighlightRow: function(row)`	Clears selected visuals
`fireRowSelectEvent: function(rowKey, behaviorEvent)`	Sends a rowSelectEvent on server side to invoke a

Function	Details
	rowSelectListener if defined
`fireRowUnselectEvent: function(rowKey, behaviorEvent)`	Sends a rowUnselectEvent on server side to invoke a rowUnselectListener if defined
`selectRowWithRadio: function(radio)`	Selects the corresping row of a radio based column selection
`unselectAllRows: function()`	
`selectAllRowsOnPage: function()`	
`unselectAllRowsOnPage: function()`	
`selectAllRows: function()`	
`toggleExpansion: function(toggler)`	Expands a row to display detail content
`collapseRow: function(row)`	
`collapseAllRows: function()`	
`getExpandedRows: function()`	
`switchToRowEdit: function(row)`	
`showRowEditors: function(row)`	
`saveRowEdit: function(rowEditor)`	Saves the edited row
`cancelRowEdit: function(rowEditor)`	
`updateRow: function(row, content)`	Updates row with given content
`clearSelection: function()`	Clears the selection state
`clearFilters: function()`	Clears table filters
`removeSelection: function(rowIndex)`	Remove given rowIndex from selection
`addSelection: function(rowKey)`	Adds given rowKey to selection if it doesn't exist already
`isSelected: function(rowKey)`	Finds if given rowKey is in selection
`saveColumnOrder: function()`	
`isEmpty: function()`	Returns if there is any data displayed
`getSelectedRowsCount: function()`	

Read widgetVar online: https://riptutorial.com/primefaces/topic/9841/widgetvar

Credits

S. No	Chapters	Contributors
1	Getting started with primefaces	Abaddon666, ArgaPK, Community, Jasper de Vries, Onur Senture, Xtreme Biker, YCF_L
2	Hello Primefaces	ArgaPK, Xtreme Biker, YCF_L
3	How to use Primefaces showcase	Skyware
4	widgetVar	Eric B., Paolo Forgia