



JavaServer™ Faces Specification

Version 1.0, Public Review Draft

February 27, 2003

Craig McClanahan and Edward Burns, editors

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303
U.S.A. 650-960-1300

JavaServer™ Faces Specification i

JSF.1.Overview	16
JSF.1.1. Solving Practical Problems of the Web	16
JSF.1.2. Specification Audience	17
JSF.1.2.1. Page Authors	17
JSF.1.2.2. Component Writers	18
JSF.1.2.3. Application Developers	19
JSF.1.2.4. Tool Providers	19
JSF.1.2.5. JSF Implementors	20
JSF.1.3. Introduction to JSF APIs	20
JSF.1.4. Using the JSF APIs	20
JSF.2.Request Processing Lifecycle	22
JSF.2.1. Request Processing Lifecycle Scenarios	23
JSF.2.1.1. Non-Faces Request Generates Faces Response	23
JSF.2.1.2. Faces Request Generates Faces Response	23
JSF.2.1.3. Faces Request Generates Non-Faces Response	24
JSF.2.2. Standard Request Processing Lifecycle Phases	24
JSF.2.2.1. Reconstitute Component Tree	24
JSF.2.2.2. Apply Request Values	25
JSF.2.2.3. Process Validations	26
JSF.2.2.4. Update Model Values	26
JSF.2.2.5. Invoke Application	27
JSF.2.2.6. Render Response	28
JSF.2.3. Common Event Processing	29
JSF.2.4. Common Application Activities	30
JSF.2.4.1. Acquire Faces Object References	30
JSF.2.4.1.1. Acquire and Configure Lifecycle Reference	30
JSF.2.4.1.2. Acquire and Configure FacesContext Reference	31
JSF.2.4.2. Create And Configure A New Component Tree	31
JSF.2.4.2.1. Create A New Component Tree	32
JSF.2.4.2.2. Configure the Desired RenderKit	32
JSF.2.4.2.3. Switch To The New Tree	33
JSF.2.4.2.4. Configure The Tree's Components	33
JSF.3.User Interface Component Model	34
JSF.3.1. UIComponent and UIComponentBase	35
JSF.3.1.1. Component Type	35
JSF.3.1.2. Component Identifiers	35
JSF.3.1.3. Client Identifiers	36

JSF.3.1.4.	Component Tree Manipulation	36
JSF.3.1.5.	Component Tree Navigation.....	37
JSF.3.1.6.	Facet Management.....	37
JSF.3.1.7.	Generic Attributes.....	38
JSF.3.1.8.	Render-Independent Properties.....	39
JSF.3.1.9.	Component Specialization Methods	40
JSF.3.1.10.	Lifecycle Management Methods	42
JSF.3.2.	NamingContainer and UINamingContainer	43
JSF.3.3.	Conversion Model.....	43
JSF.3.3.1.	Overview.....	43
JSF.3.3.2.	Converter	44
JSF.3.3.3.	ConverterFactory	45
JSF.3.3.4.	Standard Converter Implementations	46
JSF.3.4.	Event and Listener Model	47
JSF.3.4.1.	Overview.....	47
JSF.3.4.2.	Event Classes	48
JSF.3.4.3.	Listener Classes	48
JSF.3.4.4.	Phase Identifiers.....	49
JSF.3.4.5.	Listener Registration	49
JSF.3.4.6.	Event Queueing.....	49
JSF.3.4.7.	Event Broadcasting	50
JSF.3.4.8.	Application Events.....	51
JSF.3.5.	Validation Model	51
JSF.3.5.1.	Overview.....	51
JSF.3.5.2.	Validator Classes	51
JSF.3.5.3.	Validation Registration	52
JSF.3.5.4.	Validation Processing	52
JSF.3.5.5.	Standard Validator Implementations	53
JSF.4.	Standard User Interface Components.....	56
JSF.4.1.	UICommand.....	57
JSF.4.2.	UIForm.....	57
JSF.4.3.	UIGraphic	58
JSF.4.4.	UIInput	58
JSF.4.5.	UIOutput	59
JSF.4.6.	UIPanel	59
JSF.4.7.	UIParameter	60
JSF.4.8.	UISelectBoolean	60
JSF.4.9.	SelectItem	61

JSF.4.10. UISelectItem	61
JSF.4.11. UISelectItems.....	61
JSF.4.12. UISelectMany	62
JSF.4.13. UISelectOne	62
JSF.5.Per-Request State Information	64
JSF.5.1. FacesContext.....	64
JSF.5.1.1. Servlet API Components	64
JSF.5.1.2. Locale.....	65
JSF.5.1.3. Component Tree	65
JSF.5.1.4. Application Events Queue	65
JSF.5.1.5. Message Queues	66
JSF.5.1.6. Lifecycle Management Objects	66
JSF.5.1.7. Model Reference Expression Evaluation.....	66
JSF.5.1.8. FacesEvents Queue	67
JSF.5.1.9. ResponseStream and ResponseWriter	67
JSF.5.1.10. Flow Control Methods	67
JSF.5.1.11. Access To The Current FacesContext Instance	68
JSF.5.2. Message	68
JSF.5.3. MessageImpl.....	69
JSF.5.4. ResponseStream.....	69
JSF.5.5. ResponseWriter.....	69
JSF.5.6. MessageResources	71
JSF.5.7. MessageResourcesFactory	72
JSF.5.8. FacesContextFactory	73
JSF.5.9. Tree	73
JSF.5.10. TreeFactory	74
JSF.6.Lifecycle Management	76
JSF.6.1. Lifecycle	76
JSF.6.2. Phase	77
JSF.6.3. ApplicationHandler.....	78
JSF.6.4. ViewHandler.....	79
JSF.6.5. LifecycleFactory	79
JSF.7.Rendering Model.....	80
JSF.7.1. RenderKit.....	80
JSF.7.2. Renderer.....	81
JSF.7.3. RenderKitFactory.....	83
JSF.7.4. Standard HTML RenderKit Implementation	83
JSF.7.4.1. Renderer Types for UICommand Components	85

JSF.7.4.2.	Renderer Types for UIForm Components	87
JSF.7.4.3.	Renderer Types for UIGraphic Components	88
JSF.7.4.4.	Renderer Types for UIInput Components	89
JSF.7.4.5.	Renderer Types for UIOutput Components.....	91
JSF.7.4.6.	Renderer Types for UIPanel Components	93
JSF.7.4.7.	Renderer Types for UISelectBoolean Components	94
JSF.7.4.8.	Renderer Types for UISelectMany Components..	95
JSF.7.4.9.	Renderer Types for UISelectOne Components	96
JSF.8.Integration With JSP	98	
JSF.8.1.	UIComponent Custom Actions.....	98
JSF.8.2.	Using UIComponent Custom Actions in JSP Pages.	100
JSF.8.2.1.	Declaring the Custom Actions Tag Library.....	100
JSF.8.2.2.	Defining Components in a Page	100
JSF.8.2.3.	Creating Components and Overriding Attributes	101
JSF.8.2.4.	Representing Component Hierarchies	101
JSF.8.2.5.	Registering Event Listeners and Validators.....	102
JSF.8.2.6.	Using Facets.....	103
JSF.8.2.7.	Interoperability with Other Custom Action Libraries	104
JSF.8.2.8.	Composing Pages from Multiple Sources	105
JSF.8.3.	The <f:use_faces/> Tag	106
JSF.8.4.	UIComponent Custom Action Implementation Requirements	106
JSF.8.5.	JSF Core Tag Library	106
JSF.8.5.1.	<f:action_listener>	108
JSF.8.5.2.	<f:attribute>	109
JSF.8.5.3.	<f:facet>.....	110
JSF.8.5.4.	<f:parameter>.....	111
JSF.8.5.5.	<f:use_faces>	113
JSF.8.5.6.	<f:validate_doublerange>	114
JSF.8.5.7.	<f:validate_length>	115
JSF.8.5.8.	<f:validate_longrange>.....	116
JSF.8.5.9.	<f:validate_required>.....	117
JSF.8.5.10.	<f:validate_stringrange>.....	118
JSF.8.5.11.	<f:validator>	119
JSF.8.5.12.	<f:valuechanged_listener>.....	120
JSF.8.6.	Standard HTML RenderKit Tag Library	121
JSF.9.Using JSF In Web Applications	122	
JSF.9.1.	Web Application Deployment Descriptor	122

JSF.9.1.1. Servlet Definition.....	123
JSF.9.1.2. Servlet Mapping.....	123
JSF.9.1.3. Application Configuration Parameters	123
JSF.9.2. Included Classes and Resources	124
JSF.9.2.1. Application-Specific Classes and Resources	124
JSF.9.2.2. Servlet and JSP API Classes (javax.servlet.*)....	124
JSF.9.2.3. JavaServer Faces API Classes (javax.faces.*)....	124
JSF.9.2.4. JavaServer Faces Implementation Classes	125
JSF.9.2.4.1. FactoryFinder.....	125
JSF.9.2.4.2. FacesServlet	126
JSF.9.2.4.3. FacesTag	127
JSF.9.2.4.4. FacesBodyTag	127
JSF.9.3. Included Configuration Files	127
JSF.10. JavaServer Faces API	128
JSF.10.1. Package javax.faces	128
JSF.10.2. Package javax.faces.component	136
JSF.10.3. Package javax.faces.context	250
JSF.10.4. Package javax.faces.convert	286
JSF.10.5. Package javax.faces.event.....	296
JSF.10.6. Package javax.faces.lifecycle	322
JSF.10.7. Package javax.faces.render	337
JSF.10.8. Package javax.faces.tree	354
JSF.10.9. Package javax.faces.validator	359
JSF.10.10. Package javax.faces.webapp.....	394

JavaServer(TM) Faces Specification ("Specification")

Version: 1.0

Status: Pre-FCS, Public Review

Release: February 27, 2003

Copyright 2003 Sun Microsystems, Inc.

**4150 Network Circle, Santa Clara, California 95054, U.S.A
All rights reserved.**

NOTICE

The Specification is protected by copyright and the information described therein may be protected by one or more U.S. patents, foreign patents, or pending applications. Except as provided under the following license, no part of the Specification may be reproduced in any form by any means without the prior written authorization of Sun Microsystems, Inc. ("Sun") and its licensors, if any. Any use of the Specification and the information described therein will be governed by the terms and conditions of this license and the Export Control and General Terms as set forth in Sun's website Legal Terms. By viewing, downloading or otherwise copying the Specification, you agree that you have read, understood, and will comply with all of the terms and conditions set forth herein.

Subject to the terms and conditions of this license, Sun hereby grants you a fully-paid, non-exclusive, non-transferable, worldwide, limited license (without the right to sublicense) under Sun's intellectual property rights to review the Specification internally for the purposes of evaluation only. Other than this limited license, you acquire no right, title or interest in or to the Specification or any other Sun intellectual property. The Specification contains the proprietary and confidential information of Sun and may only be used in accordance with the license terms set forth herein. This license will expire ninety (90) days from the date of Release listed above and will terminate immediately without notice from Sun if you fail to comply with any provision of this license. Upon termination, you must cease use of or destroy the Specification.

TRADEMARKS

No right, title, or interest in or to any trademarks, service marks, or trade names of Sun or Sun's licensors is granted hereunder. Sun, Sun Microsystems, the Sun logo, Java, the Java Coffee Cup logo, J2EE, and JavaServer are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

DISCLAIMER OF WARRANTIES

THE SPECIFICATION IS PROVIDED "AS IS" AND IS EXPERIMENTAL AND MAY CONTAIN DEFECTS OR DEFICIENCIES WHICH CANNOT OR WILL NOT BE CORRECTED BY SUN. SUN MAKES NO REPRESENTATIONS OR WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT THAT THE CONTENTS OF THE SPECIFICATION ARE SUITABLE FOR ANY PURPOSE OR THAT ANY PRACTICE OR IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADE SECRETS OR OTHER RIGHTS. This document does not represent any commitment to release or implement any portion of the Specification in any product.

THE SPECIFICATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION THEREIN; THESE CHANGES WILL BE INCORPORATED INTO NEW VERSIONS OF THE SPECIFICATION, IF ANY. SUN MAY MAKE IMPROVEMENTS AND/OR CHANGES TO THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THE SPECIFICATION AT ANY TIME. Any use of such changes in the Specification will be governed by the then-current license for the applicable version of the Specification.

LIMITATION OF LIABILITY

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION, LOST REVENUE, PROFITS OR DATA, OR FOR SPECIAL, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF OR RELATED TO ANY FURNISHING, PRACTICING, MODIFYING OR ANY USE OF THE SPECIFICATION, EVEN IF SUN AND/OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You will indemnify, hold harmless, and defend Sun and its licensors from any claims based on your use of the Specification for any purposes other than those of internal evaluation, and from any claims that later versions or releases of any Specification furnished to you are incompatible with the Specification provided to you under this license.

RESTRICTED RIGHTS LEGEND

If this Software is being acquired by or on behalf of the U.S. Government or by a U.S. Government prime contractor or subcontractor (at any tier), then the Government's rights in the Software and accompanying documentation shall be only as set forth in this license; this is in accordance with 48 C.F.R. 227.7201 through 227.7202-4 (for Department of Defense (DoD) acquisitions) and with 48 C.F.R. 2.101 and 12.212 (for non-DoD acquisitions).

REPORT

You may wish to report any ambiguities, inconsistencies or inaccuracies you may find in connection with your evaluation of the Specification ("Feedback"). To the extent that you provide Sun with any Feedback, you hereby: (i) agree that such Feedback is provided on a non-proprietary and non-confidential basis, and (ii) grant Sun a perpetual, non-exclusive, worldwide, fully paid-up, irrevocable license, with the right to sublicense through multiple levels of sublicensees, to incorporate, disclose, and use without limitation the Feedback for any purpose related to the Specification and future versions, implementations, and test suites thereof.

(LFI#117738/Form ID#011801)

Preface

This is the JavaServer™ Faces 1.0 (JSF 1.0) specification, developed by the JSR-127 expert group under the Java Community Process (see <<http://www.jcp.org>> for more information about the JCP).

What's changed since the last release

Changes in the Organization of this Document

In the *Request Processing Lifecycle* chapter:

- added a new section: *Request Processing Lifecycle Scenarios*.
- Grouped the standard lifecycle phases into the section *Standard Request Processing Lifecycle Phases*
- Introduce new sections on *Common Event Processing* and *Common Application Activities*.

In the *User Interface Component Model* chapter

- Extensive reorganization. Added sections on *Facets*, *Render-Independent Properties*, *Component Specialization Methods*, *Lifecycle Management Methods*, *Naming Containers*, *Conversion Model*, and *Validation Model*.
- Added section *Event and Listener Model*

In the *Standard User Interface Components* chapter:

- We still have a sub-section for each component, but we have some new components.

In the *Per-Request State Information* chapter:

- Change sub-section *Request Events Queue* to *FacesEvents Queue*
- Added sub-sections: *Flow Control Methods*, *Access to To The Current FacesContext Instance*, *Tree and TreeFactory*.

In the *Rendering Model* chapter, we fleshed out the sub-section: *Standard HTML RenderKit Implementation*.

In the *Integration with JSP* chapter, added sub-sections: *Using Facets*, *JSF Core Tag Library*, *Standard HTML RenderKit Tag Library*

Added chapter *JavaServer Faces API*.

New Event Model

New Event model as in section JSF.3.3

This is the biggest change since the previous release. This change is too big to cover in detail in the README. Two important aspects are: we now have "ValueChangedEvents" and there is only one tree, not separate request and response trees as before. Please read the appropriate spec section.

Flat namespace changes

The compoundId property has gone away, in its place we have getClientId(). Component identifiers are no longer required in all cases. When using JSF from JSP, component identifiers are only required when interacting with JSTL. See Section JSF 3.1.2 in the spec. We have a tag library validator to enforce this rule.

Every component can be named by a component identifier, which must be unique among the components that share a common naming container parent in a component tree.

The root of the component tree must be a NamingContainer.

Support for component facets.

An additional useful facility would be to define particular subordinate components that have a specific role with respect to the owning component, which is typically independent of the parent-child relationship. An example might be a "data grid" control, where the children represent the columns to be rendered in the grid. It is useful to be able to identify a component that represents the column header and/or footer, separate from the usual child collection that represents

the column data. To meet this requirement, JavaServer Faces components offer support for facets, which represent a named collection of subordinate (but non-child) components that are related to the current component by virtue of a unique facet name that represents the role that particular component plays. See section JSF 3.1.6 for more info.

Traversal Order

Change the way components are processed so they honor the new Facet work. In addition, add new UIComponent traversal methods.

Instead of using the TreeNavigator class to traverse the tree, this is now handled by the UIComponent implementations.

Visible Property

Add support for the "rendered" property, which is leveraged during encode/decode. See section JSF 3.1.12 of the spec.

New Tag Library

The URI for the html basic tld has changed to be spec compliant. It is now <<http://java.sun.com/jsf/html>>. The recommended prefix for this tag lib is "h".

We have broken the non-component tags out into the jsf-core taglib. The URI for this taglib is <<http://java.sun.com/jsf/core>>. The recommended prefix for this tag lib is "f".

Name changes

Class javax.faces.component.UIInput replaces javax.faces.component.UITextEntry. UIInput is for anything that can take input from the user. This change impacts the tags as well. For example, input_secret takes the place of textentry_secret.

selectone_radio takes the place of selectone_radiogroup

selectone_listbox takes the place of selectone_optionlist

output_errors takes the place of validation_message

UIParameter

Class javax.faces.component.UIParameter and the accompanying tag are used to specify parameters to components. For example, in the case of output_message:

```
<h:output_message id="userMsg"
    value="Welcome {0}. Thanks for trying the {1}
    application." >

    <f:parameter id="param1"
    modelReference="LoginBean.userName" />

    <f:parameter id="param2" value="Faces Basic"/>

</h:output_message>
```

Converters

Add support for type conversion as in section JSF.3.2. Converters enable model data to be stored in a model specific type, such as java.util.Date even though. In addition, the complete set of rich formatting offered by the java.text package is exposed through Converters.

Standard Renderkit

Several new Renderers, and accompanying tags, have been added. Please see Section JSF.7.4 for details.

Validators API

Please see section JSF.2.2.3 for more detail. There are now tags for the commonly used validators.

Deprecated APIs

```
javax.faces.context.FacesContext.addApplicationEvent()
```

```
javax.faces.context.FacesContext.getApplicationEvents()
javax.faces.context.FacesContext.getApplicationEventsCount()
javax.faces.context.FacesContext.getApplicationHandler()
javax.faces.event.CommandEvent
javax.faces.event.FormEvent
javax.faces.lifecycle.ApplicationHandler
javax.faces.lifecycle.Lifecycle.getApplicationHandler()
javax.faces.lifecycle.Lifecycle.setApplicationHandler()
javax.faces.lifecycle.Lifecycle.executePhase()
javax.faces.lifecycle.Phase
```

Other Java™ Platform Specifications

JSF is based on the following Java API specifications:

- JavaServer Pages Specification, version 1.2 (JSP™)
<http://java.sun.com/products/jsp/>
- Java Servlet Specification, version 2.3 (Servlet)
<http://java.sun.com/products/servlet/>
- Java 2 Platform, Standard Edition, version 1.3
<http://java.sun.com/j2se/>
- JavaBeans Specification, version 1.0.1
<http://java.sun.com/products/javabeans/docs/spec.html>

In addition, JSF is designed to work synergistically with other web-related Java APIs, including:

- JavaServer Pages Standard Tag Library, version 1.0 (JSTL)
<http://java.sun.com/products/jsp/jstl/>
- Portlet Specification, under development in JSR-168
<http://www.jcp.org/jsr/detail/168.jsp>

Related Documents and Specifications

The following documents and specifications of the World Wide Web Consortium will be of interest to JSF implementors, as well as developers of applications and components based on JavaServer Faces.

- Hypertext Markup Language (HTML), version 4.01
<http://www.w3.org/TR/html4/>

- Extensible HyperText Markup Language (XHTML), version 1.0
<http://www.w3.org/TR/xhtml1>
- Extensible Markup Language (XML), version 1.0 (Second Edition)
<http://www.w3.org/TR/REC-xml>
- XForms 1.0 (currently a Candidate Recommendation)
<http://www.w3.org/TR/xforms/>

Providing Feedback

We welcome any and all feedback about this specification. Please email your comments to <jsr127-comments@sun.com>.

Please note that, due to the volume of feedback that we receive, you will not normally receive a reply from an engineer. However, each and every comment is read, evaluated, and archived by the specification team.

Acknowledgements

The JavaServer Faces specification is the result of collaborative work involving many individuals on the JSR-127 expert group ... (FIXME - individual acknowledgements as needed)

JSF.1

Overview

JavaServer Faces (JSF) is a *user interface* (UI) framework for Java Web applications. It is designed to significantly ease the burden of writing and maintaining applications which run on a Java application server, and render their UIs back to a target client. JSF provides ease-of-use in the following ways:

- Makes it easy to construct a UI from a set of reusable UI components
- Simplifies migration of application data to and from the UI
- Helps manage UI state across server requests
- Provides a simple model for wiring client-generated events to server-side application code
- Allows custom UI components to be easily built and re-used

Most importantly, JSF establishes standards which are designed to be leveraged by tools to provide a developer experience which is accessible to a wide variety of developer types, ranging from corporate developers to systems programmers. A “corporate developer” is characterized as an individual who is proficient in writing procedural code and business logic, but is not necessarily skilled in object-oriented programming. A “systems programmer” understands object-oriented fundamentals, including abstraction and designing for re-use. A corporate developer typically relies on tools for development, while a system programmer may define his or her tool as a text editor for writing code.

Therefore, JSF is designed to be tooled, but also exposes the framework and programming model as APIs so that it can be used outside of tools, as is sometimes required by systems programmers.

JSF.1.1 Solving Practical Problems of the Web

JSF’s core architecture is designed to be independent of specific protocols and markup; however it is aimed directly at solving many of the common problems encountered when writing applications for HTML clients which communicate via HTTP to a Java application server that supports servlets and JavaServer Pages (JSP)

based applications. These applications are typically form-based, comprised of one or more HTML pages which the user interacts with to complete a task or set of tasks. JSF tackles the following challenges associated with these applications:

- managing UI component state across requests
- supporting encapsulation of the differences in markup across different browsers and clients
- supporting form processing (multi-page, more than one per page, etc)
- providing a strongly typed event model which allows the application to write server-side handlers (independent of HTTP) for client generated events
- validating request data and providing appropriate error reporting
- enabling type conversion when migrating component values (Strings) to/from application data objects (often not Strings)
- handling error and exceptions, and reporting errors in human-readable form back to the application user

JSF.1.2 Specification Audience

The JSF specification, and the technology that it defines, is addressed to several audiences that will use this information in different ways. The following sections describe these audiences, the roles that they play with respect to JSF, and how they will use the information contained in this document. As is the case with many technologies, the same person may play more than one of these roles in a particular development scenario; however, it is still useful to understand the individual viewpoints separately.

JSF.1.2.1 Page Authors

A *page author* is primarily responsible for creating the user interface of a web application. He or she must be familiar with the markup language(s) (such as HTML and JavaScript) that are understood by the target client devices, as well as the rendering technology (such as JavaServer Pages) used to create dynamic content. Page authors are often focused on graphical design and human factors engineering, and are generally not familiar with programming languages such as Java or Visual Basic (although many page authors will have a basic understanding of client side scripting languages such as JavaScript).

From the perspective of JSF, page authors will generally assemble the content of the pages being created from libraries of prebuilt user interface components that are provided by component writers, tool providers, and JSF implementors. The components themselves will be represented as configurable objects that utilize the dynamic markup capabilities of the underlying rendering technology. When JavaServer Pages are in use, for example, components will be represented as JSP custom actions, which will support configuring the attributes of those components

as custom action attributes in the JSP page. In addition, the pages produced by a page author will be used by the JSF framework to create component tree hierarchies that represent the components on those pages.

Page authors will generally utilize development tools, such as HTML editors, that allow them to deal directly with the visual representation of the page being created. However, it is still feasible for a page author that is familiar with the underlying rendering technology to construct pages “by hand” using a text editor.

JSF.1.2.2 Component Writers

Component writers are responsible for creating libraries of reusable user interface objects. Such components support the following functionality:

- Convert the internal representation of the component’s properties and attributes into the appropriate markup language for pages being rendered (encoding).
- Convert the properties of an incoming request -- parameters, headers, and cookies -- into the corresponding properties and attributes of the component (decoding)
- Utilize request-time events to initiate visual changes in one or more components, followed by redisplay of the current page.
- Support validation checks on the syntax and semantics of the representation of this component on an incoming request, as well as conversion into the internal form that is appropriate for this component.

As discussed in Chapter JSF.7 “Rendering Model, the encoding and decoding functionality may optionally be delegated to one or more *Render Kits*, which are responsible for customizing these operations to the precise requirements of the client that is initiating a particular request (for example, adapting to the differences between JavaScript handling in different browsers, or variations in the WML markup supported by different wireless clients).

The component writer role is sometimes separate from other JSF roles, but is often combined. For example, reusable components, component libraries, and render kits might be created by:

- A page author creating a custom “widget” for use on a particular page
- An application developer providing components that correspond to specific data objects in the application’s business domain
- A specialized team within a larger development group responsible for creating standardized components for reuse across applications
- Third party library and framework providers creating component libraries that are portable across JSF implementations
- Tool providers whose tools can leverage the specific capabilities of those libraries in development of JSF-based applications

- JSF implementors who provide implementation-specific component libraries as part of their JSF product suite

Within JSF, user interface components are represented as Java classes that follow the design patterns outlined in the JavaBeans Specification. Therefore, new and existing tools that facilitate JavaBean development can be leveraged to create new JSF components. In addition, the fundamental component APIs are simple enough for developers with basic Java programming skills to program by hand.

JSF 1.2.3 Application Developers

Application Developers are responsible for providing the server side functionality of a web application that is not directly related to the user interface. This encompasses following general areas of responsibility:

- Define mechanisms for persistent storage of the information required by JSF-based web applications (such as creating schemas in a relational database management system)
- Create a Java object representation of the persistent information, such as Entity Enterprise JavaBeans (Entity EJBs), and call the corresponding beans as necessary to perform persistence of the application's data.
- Encapsulate the application's functionality, or business logic, in Java objects that are reusable in web and non-web applications, such as Session EJBs.
- Expose the data representation and functional logic objects for use via JSF, as would be done for any servlet or JSP based application.

Only the latter responsibility is directly related to JavaServer Faces APIs. In particular, the following steps are required to fulfill this responsibility:

- Expose the underlying data required by the user interface layer as objects that are accessible from the web tier (such as via request or session attributes in the Servlet API), via *model reference expressions*, as described in Chapter JSF.3 “Render-Independent Properties”
- Provide application-level event handlers for the events that are enqueued by JSF components during the request processing lifecycle, as described in Section JSF.2.2.5 “Invoke Application”

Application modules interact with JSF through standard Java APIs, and can therefore be created using new and existing tools that facilitate general Java development. In addition, application modules can be written (either by hand, or by being generated) in conformance to an application framework created by a tool provider.

JSF.1.2.4 Tool Providers

Tool providers, as their name implies, are responsible for creating tools that assist in the development of JSF-based applications, rather than creating such applications directly. JSF APIs support the creation of a rich variety of development tools, which can create applications that are portable across multiple JSF implementations.

Examples of possible tools include:

- GUI-oriented page development tools that assist page authors in creating the user interface for a web application
- IDEs that facilitate the creation of components (either for a particular page, or for a reusable component library)
- Page generators that work from a high level description of the desired user interface to create the corresponding page and component objects
- IDEs that support the development of general web applications, adapted to provide specialized support (such as configuration management) for JSF
- Web application frameworks (such as MVC-based and workflow management systems) that facilitate the use of JSF components for user interface design, in conjunction with higher level navigation management and other services
- Application generators that convert high level descriptions of an entire application into the set of pages, UI components, and application modules needed to provide the required application functionality

Tool providers will generally leverage the JSF APIs for introspection of the features of component libraries and render kit frameworks, as well as the application portability implied by the use of standard APIs in the code generated for an application.

JSF.1.2.5 JSF Implementors

Finally, *JSF implementors* will provide runtime environments that implement all of the requirements described in this specification. Typically, a JSF implementor will be the provider of a Java2 Enterprise Edition (J2EE) application server, although it is also possible to provide a JSF implementation that is portable across J2EE servers.

Advanced features of the JSF APIs allow JSF implementors, as well as application developers, to customize and extend the basic functionality of JSF in a portable way. These features provide a rich environment for server vendors to compete on features and quality of service aspects of their implementations, while maximizing the portability of JSF-based applications across different JSF implementations.

JSF.1.3 Introduction to JSF APIs

This section briefly describes major functional subdivisions of the APIs defined by JavaServer Faces. Each subdivision is described by its own chapter, later in this specification.

FIXME - add summary of each package here.

JSF.1.4 Using the JSF APIs

FIXME - some simple examples of using JSF.

JSF.2

Request Processing Lifecycle

Each servlet request that involves a JSF component tree goes through a well-defined *request processing lifecycle* made up of *phases*. There are three different scenarios that must be considered, each with its own combination of phases and activities:

- Non-Faces Request Generates Faces Response
- Faces Request Generates Faces Response
- Faces Request Generates Non-Faces Response

Where the terms being used are defined as follows:

- *Faces Response* -- A servlet response that was created by the execution of the *Render Response* phase of the request processing lifecycle.
- *Non-Faces Response* -- A servlet response that was not created by the execution of the *Render Response* phase of the request processing lifecycle. Examples would be a servlet-generated or JSP-rendered response that does not incorporate JSF components, or a response that sets an HTTP status code other than the usual 200 (such as a redirect).
- *Faces Request* -- A servlet request that was sent from a previously generated *Faces Response*. Examples would be a hyperlink or form submit from a rendered user interface component, where the request URI was crafted (by the component or renderer that created it) to identify the component tree to use for processing the request.
- *Non-Faces Request* -- A servlet request that was sent to an application component (e.g. a servlet or JSP page), rather than directed to a Faces component tree.

In addition, of course, your web application may receive Non-Faces Requests that generate Non-Faces Responses. Because such requests do not involve JavaServer Faces at all, their processing is outside the scope of this specification, and will not be considered further.

READER NOTE: The dynamic behavior descriptions in this Chapter make forward references to the Sections that describe the individual classes and interfaces. You will probably find it useful to follow the reference and skim the definition of each new class or interface as you encounter them, then come back and finish the behavior description. Later, you can study the characteristics of each JSF API in the subsequent chapters.

JSF.2.1 Request Processing Lifecycle Scenarios

Each of the scenarios described above has a lifecycle that is composed of a particular set of phases, executed in a particular order. The scenarios are described individually in the following subsections.

JSF.2.1.1 Non-Faces Request Generates Faces Response

An application that is processing a Non-Faces Request may desire to utilize JSF to render a Faces Response to that request. In order to accomplish this, the application must perform the common activities that are described in the following sections:

- Acquire Faces Object References, as described in Section [FIXME/2.4.1](#), below.
- Create A New Component Tree, as described in Section [FIXME/2.4.2](#), below.
- Call `renderResponse()` on the `FacesContext` instance that was acquired. This signals the JSF implementation that processing should immediately proceed to the *Render Response* phase of the request processing lifecycle.
- Call the `execute()` method on the `Lifecycle` instance that was acquired. This signals the JSF implementation to begin processing at the next appropriate phase of the request processing lifecycle. Because of the call to `renderResponse()` in the previous step, control will proceed immediately to rendering.

JSF.2.1.2 Faces Request Generates Faces Response

The most common lifecycle will be the case where a previous Faces Response includes user interface controls that will submit a subsequent request to this web application, utilizing a request URI that is mapped to the JSF implementation's controller, as described in Section [JSF.9.1.2 "Servlet Mapping"](#). Because such a request will be initially handled by the JSF implementation, the application need not take any special steps -- its event listeners, validators, and application handler will be invoked at appropriate times as the standard request processing lifecycle, described in the following diagram, is invoked.

[ED. NOTE -- Revised diagram goes here]

The behavior of the individual phases of the request processing lifecycle are described in individual subsections of Section JSF.2.2 “Standard Request Processing Lifecycle Phases”. Note that, at the conclusion of several phases of the request processing lifecycle, common event processing logic (as described in Section JSF.2.3 “Common Event Processing”) is performed to broadcast any `FacesEvents` generated by components in the component tree to interested event listeners.

JSF.2.1.3 **Faces Request Generates Non-Faces Response**

Normally, a JSF-based application will utilize the *Render Response* phase of the request processing lifecycle to actually create the servlet response that is sent back to the client. In some circumstances, however, this behavior might not be desirable. For example:

- A Faces Request needs to be redirected to a different web application resource (via a call to `HttpServletResponse.sendRedirect()`).
- A Faces Request causes the generation of a binary (such as an image) or text (such as XML) response instead of a response intended for human consumption (such as HTML sent back to a browser).

In any of these scenarios, the application will have used the standard mechanisms of the servlet API to create the response headers and content. What remains is the necessity to tell the JSF implementation that the response has already been created, so that the *Render Response* phase of the request processing lifecycle should be skipped. This is accomplished by calling the `responseComplete()` method on the `FacesContext` instance for the current request, prior to returning from the application handler.

JSF.2.2 **Standard Request Processing Lifecycle Phases**

The standard phases of the request processing lifecycle are described in the following subsections.

JSF.2.2.1 **Reconstitute Component Tree**

The JSF implementation must perform the following tasks during the *Reconstitute Component Tree* phase of the request processing lifecycle:

- Acquire, from the incoming request and/or saved information on the server, the state information required to construct the component tree. In order to be utilized, such state information must meet the following requirements:

- It must have a tree identifier that corresponds to the request URI used to submit this request. In other words, if the JSF implementation has saved state information for a tree identifier `/foo` in the user's session, but receives a request with URI `/faces/bar`, the saved state information will be thrown away.
- If such state information is available, use it to construct a component Tree, including any required wire-up of event listeners and validators, and setting the appropriate RenderKit instance. Save the created Tree instance in the FacesContext for the current request, by passing it as a parameter to the `setTree()` method.
- If no such state information is available, or if any saved state information was for a different page than the one processing this request, acquire a component Tree for the requested tree identifier, by calling the `getTree()` method of the TreeFactory instance for this web application, and store it in the FacesContext by calling `setTree()`. The tree identifier for this tree is extracted from the *extra path info* portion of the request URI. This means that, when using the standard servlet mapping of `/faces/*` to `javax.faces.webapp.FacesServlet`, a request URI `/faces/menu.jsp` will select tree identifier `/menu.jsp`.
- Call the `setLocale()` method of the FacesContext instance for the current request, passing a Locale instance derived from the saved state information (if any); otherwise, acquire the default Locale to be used by calling the `getLocale()` method of the ServletRequest for this request.

At the end of this phase, the `tree` property of the FacesContext instance for the current request will reflect the saved configuration of the component tree generated by the previous Faces Response (if any).

JSF 2.2.2 Apply Request Values

The purpose of the *Apply Request Values* phase of the request processing lifecycle is to give each component the opportunity to update its current value from the information included in the current request (parameters, headers, cookies, and so on).

During the *Apply Request Values* phase, the JSF implementation must call the `processDecodes()` method of the root component in the component tree. This will normally cause the `processDecodes()` method of each component in the tree to be called recursively, as described in the Javadocs for the `UIComponent.processDecodes()` method.

During the decoding of component values, events may have been queued by the components and/or renderers whose `decode()` method was invoked. Perform the *Common Event Processing*, described in Section JSF 2.3 “Common Event Processing”, to ensure that all such events are broadcast to all interested event listeners.

At the end of this phase, all components in the component tree will have been updated with new values included in this request (or enough data to reproduce incorrect input will have been stored, if there were conversion errors). Conversions that failed will have caused messages to be enqueued via calls to the `addMessage()` method of the `FacesContext` instance for the current request, and the `valid` property on the corresponding component(s) will have been set to `false`.

If any of the `decode()` methods that was invoked, or an event listener that processed a queued event, called `responseComplete()` on the `FacesContext` instance for the current request, lifecycle processing of the current request must be immediately terminated. If any of the `decode()` methods that was invoked, or an event listener that processed a queued event, called `renderResponse()` on the `FacesContext` instance for the current request, control must be transferred to the *Render Response* phase of the request processing lifecycle. Otherwise, control must proceed to the *Process Validations* phase.

JSF.2.2.3 Process Validations

As part of the creation of the component tree for this request, zero or more `Validator` instances may have been registered for each component. In addition, component classes themselves may implement validation logic in their `validate()` methods.

During the *Process Validations* phase of the request processing lifecycle, the JSF implementation must call the `processValidators()` method of the root component in the component tree. This will normally cause the `processValidators()` method of each component in the tree to be called recursively, as described in the Javadocs for the `UIComponent.processValidators()` method.

During the processing of validations, events may have been queued by the components and/or Validators whose `validate()` method was invoked. Perform the *Common Event Processing*, described in Section JSF.2.3 “Common Event Processing”, to ensure that all such events are broadcast to all interested event listeners.

At the end of this phase, all configured validations will have been completed. Validations that failed will have caused messages to be enqueued via calls to the `addMessage()` method of the `FacesContext` instance for the current request, and the `valid` property on the corresponding components will have been set to `false`.

If any of the `validate()` methods that was invoked, or an event listener that processed a queued event, called `responseComplete()` on the `FacesContext` instance for the current request, lifecycle processing of the current request must be immediately terminated. If any of the `validate()` methods that was invoked, or an event listener that processed a queued event, called `renderResponse()` on the `FacesContext` instance for the current request, control must be transferred to the

Render Response phase of the request processing lifecycle. If there is at least one message queued via a call to the `addMessage()` method on the `FacesContext` instance for the current request (indicating a conversion failure or a validation failure), control must be transferred to the *Render Response* phase of the request processing lifecycle. Otherwise, control must proceed to the *Update Model Values* phase.

JSF 2.2.4 Update Model Values

If this phase of the request processing lifecycle is reached, it can be assumed that the incoming request is syntactically and semantically valid (according to the validations that were performed), that the local value of every component in the component tree has been updated, and that it is now appropriate to update the application's model data in preparation for performing any application events that have been enqueued.

During the *Update Model Values* phase, the JSF implementation must call the `processUpdates()` method of the root component in the component tree. This will normally cause the `processUpdates()` method of each component in the tree to be called recursively, as described in the Javadocs for the `UIComponent.processUpdates()` method. The actual model update for a particular component is done in the `updateModel()` method for that component.

During the processing of model updates, events may have been queued by the components whose `updateModel()` method was invoked. Perform the *Common Event Processing*, described in Section JSF 2.3 "Common Event Processing", to ensure that all such events are broadcast to all interested event listeners.

At the end of this phase, all appropriate model data objects will have had their values updated to match the local value of the corresponding component, and the component local values will have been cleared.

If any of the `updateModel()` methods that was invoked, or an event listener that processed a queued event, called `responseComplete()` on the `FacesContext` instance for the current request, lifecycle processing of the current request must be immediately terminated. If any of the `updateModel()` methods that was invoked, or an event listener that processed a queued event, called `renderResponse()` on the `FacesContext` instance for the current request, control must be transferred to the *Render Response* phase of the request processing lifecycle. Otherwise, control must proceed to the *Invoke Application* phase.

JSF 2.2.5 Invoke Application

[ED. NOTE -- The behavior of this phase will be revised in a future version of this specification. The current mechanism is a placeholder until a more sophisticated interface design is completed.]

The JSF implementation must perform the following tasks during the *Invoke Application* phase of the request processing lifecycle:

- If no application events have been queued for this request, take no further action.
- Call the `getApplicationHandler()` method of the `FacesContext` instance for the current request, to get a reference to the registered application event handler (see section [FIXME/6.3](#) for more information). If no such handler has been registered, throw a `FacesException`.
- Call the `getApplicationEvents()` method on the `FacesContext` instance for the current request, and dispatch each queued event to the `processEvent()` method on the application event handler.
- If the `processEvent()` method of the application event handler returns `true`, return `Phase.GOTO_RENDER` in order to proceed immediately to the *Render Response* phase. Otherwise, continue event handling until all queued application events have been processed.

Application event handlers can perform whatever application-level functions are appropriate to deal with the event(s) that have been dispatched. However, application event handlers may perform the following operations that are directly relevant to JSF request lifecycle processing:

- Create a new component tree for the response, instead of reusing the one associated with the current request. See Section [JSF.2.4.2 “Create And Configure A New Component Tree”](#) for details.
- Add and remove components in the component tree, and/or customize their attributes.
- Change the `RenderKit` that will be utilized during the *Render Response* phase for the current response.
- Enqueue error messages by calling an appropriate `addMessage()` method on the `FacesContext`.
- Enqueue one or more additional application events to be handled during this phase.

If the application event handler called `responseComplete()` on the `FacesContext` instance for the current request, lifecycle processing of the current request must be immediately terminated. Otherwise, control must proceed to the *Invoke Application* phase.

JSF.2.2.6 Render Response

JSF supports a range of approaches that JSF implementations may utilize in creating the response text that corresponds to the contents of the response component tree, including:

- Deriving all of the response content directly from the results of the encoding methods (on either the components or the corresponding renderers) that are called.
- Interleaving the results of component encoding with content that is dynamically generated by application programming logic.
- Interleaving the results of component encoding with content that is copied from a static “template” resource.
- Interleaving the results of component encoding by embedding calls to the encoding methods into a dynamic resource (such as representing the components as custom tags in a JSP page).

Because of the number of possible options, the mechanism for implementing the *Render Response* phase cannot be specified precisely. However, all JSF implementations of this phase must conform to the following requirements:

- JSF implementations must provide a default `ViewHandler` implementation that performs a `RequestDispatcher.forward()` call to a web application resource whose context-relative path is derived from the tree identifier of the component tree [ED. NOTE- a more sophisticated navigation mechanism will be included in a subsequent version of this Specification].
- If all of the response content is being derived from the encoding methods of the component or associated `Renderer`s, the component tree should be walked in the same depth-first manner as was used in earlier phases to process the component tree, but subject to the additional constraints listed here.
- If the response content is being interleaved from additional sources and the encoding methods, the components may be selected for rendering in any desired order¹.
- During the rendering process, additional components may be added to the component tree based on information available to the `ViewHandler` implementation². However, before adding a new component, the `ViewHandler` implementation must check for the existence of the corresponding component in the component tree first. If the component already exists (perhaps because a previous phase has precreated one or more components), the component attributes that already exist must override any attribute settings that the `ViewHandler` attempts to make.
- Under no circumstances should a component be selected for rendering when its parent component, or any of its ancestors in the component tree, has its `rendersChildren` property set to true. In such cases, the parent or ancestor component will (or will have) rendered the content of this child component when the parent or ancestor was selected.

1. Typically, component selection will be driven by the occurrence of special markup (such as the existence of a JSP custom tag) in the template text associated with the component tree.

2. For example, this technique is used when custom tags in JSP pages are utilized as the rendering technology, as described in Chapter *FIXME/8*.

When each particular component in the component tree is selected for rendering, calls to its `encodeXXX()` methods must be performed in the manner described in Section JSF.3.1.9 “Component Specialization Methods”.

Upon completion of rendering, the completed state of the component tree must be saved in either the response being created, in the user’s session, or some combination of the above, in an implementation-defined manner. This state information must be made accessible on a subsequent request, so that the *Reconstitute Component Tree* can access it. For example, the saved state information could be encoded in an `<input type="hidden">` field inside an HTML `<form>` to be submitted by the user.

[FIXME - say something about setting response characteristics such as cookies and headers -- most particularly the content type header?]

[FIXME - suggestion that encoders do URL rewriting on hyperlinks to maintain session state?]

JSF.2.3 Common Event Processing

[ED. NOTE -- The actual implementation of the current event processing model is a little awkward, and shares responsibilities between the JSF implementation and the component’s `broadcast()` method clumsily. This needs to be reviewed and refined.]

For a complete description of the event processing model for JavaServer Faces components, see Section JSF.3.4 “Event and Listener Model”.

During several phases of the request processing lifecycle, as described in Section JSF.2.2 “Standard Request Processing Lifecycle Phases”, the possibility exists for events to be queued, which must now be broadcast to interested event listeners. The JSF implementation must take the following actions to perform this broadcast.

- Call the `getFacesEvents()` method on the `FacesContext` instance associated with the current request, to acquire an `Iterator` over the events (if any) that have been queued.
- For each queued event, call `getComponent()` to acquire a reference to the source component for that event, and call the `broadcast()` method on that component, passing the event instance and an identifier for the current phase of the request processing lifecycle.

In turn, the `broadcast()` method of each `UIComponent` class must broadcast the event to all event listeners who have registered an interest, on this source component, for events of the specified type, and then return a boolean flag indicating whether this event has been completely handled and that the JSF implementation must remove it from the event queue (by calling the `remove()` method on the `Iterator` being processed. See the JavaDocs for the `UIComponent.broadcast()` method for the detailed functional requirements.

It is also possible for event listeners to cause additional events to be enqueued for processing during the current phase of the request processing lifecycle. The `Iterator` implementation returned by `getFacesEvents()` must support dynamic modifications to the event list that is being iterated over, as described in the JavaDocs for the `getFacesEvents()` method.

JSF.2.4 Common Application Activities

The following subsections describe common activities that may be undertaken by an application that is using JSF to process an incoming request and/or create an outgoing response. Their use is described in Section JSF.2.1 “Request Processing Lifecycle Scenarios”, for each request processing lifecycle scenario in which the activity is relevant.

JSF.2.4.1 Acquire Faces Object References

This phase is only required when the servlet request being processed was not submitted from a previous response, and therefore did not initiate the *Faces Request Generates Faces Response* lifecycle. In order to generate a Faces response, the application must first acquire references to several objects provided by the JSF implementation, as follows:

JSF.2.4.1.1 Acquire and Configure Lifecycle Reference

As described in Section JSF.6.1 “Lifecycle”, the JSF implementation must provide an instance of `javax.faces.lifecycle.Lifecycle` that may be utilized to manage the remainder of the request processing lifecycle. An application may acquire a reference to this instance in a portable manner, as follows:

```
LifecycleFactory lfactory = (LifecycleFactory)
FactoryFinder.getFactory(FactoryFinder.LIFECYCLE_FACTORY);
Lifecycle lifecycle =
lfactory.getLifecycle(LifecycleFactory.DEFAULT_LIFECYCLE);
```

It is also legal to specify a different lifecycle identifier as a parameter to the `getLifecycle()` method, as long as this identifier is recognized and supported by the JSF implementation you are using. However, using a non-default lifecycle identifier will generally not be portable to any other JSF implementation.

If this is the first time you have acquired a reference to the `Lifecycle` instance for a particular lifecycle identifier, you may configure the `ApplicationHandler` that your application will provide to process application events, and/or the `ViewHandler` that the JSF implementation should use during the `renderResponse` phase of the request processing lifecycle.

If you do not define an ApplicationHandler instance, no application events will be processed during the *Invoke Application* phase of the request processing lifecycle. As described in Section JSF.6.1 “Lifecycle”, a default ViewHandler implementation is used if you do not specify a different one.

JSF.2.4.1.2 Acquire and Configure FacesContext Reference

[ED. NOTE -- the details of the following configuration will change in a subsequent version of this specification.]

As described in Section JSF.5.1 “FacesContext”, the JSF implementation must provide an instance of `javax.faces.context.FacesContext` to contain all of the per-request state information for a Faces Request or a Faces Response. An application that is processing a Non-Faces Request, but wanting to create a Faces Response, must acquire a reference to a `FacesContext` instance as follows:

```
FacesContextFactory fcFactory = (FacesContextFactory)
FactoryFinder.getFactory(FactoryFinder.FACES_CONTEXT_FACTORY);
FacesContext facesContext =
fcFactory.getFacesContext(context, request, response, lifecycle);
```

JSF.2.4.2 Create And Configure A New Component Tree

When a Faces Response is being intially created, or when the application decides it wants to create and configure a new component tree that will ultimately be rendered, it may follow the steps described below in order to set up the component tree that will be used. You must start with a reference to a `FacesContext` instance for the current request.

[FIXME -- consider abstracting the nitty gritty details into easier-to-user utility methods.]

JSF.2.4.2.1 Create A New Component Tree

As described in Section JSF.5.9 “Tree”, component trees are represented by a data structure rooted in an instance of `javax.faces.tree.Tree`, and identified by a tree identifier whose meaning depends on the ViewHandler implementation to be used during the *Render Response* phase of the request processing lifecycle¹. In addition, it provides a `TreeFactory` that may be utilized to construct new component trees, as follows:

```
String treeId = ... identifier of the desired Tree ...;
TreeFactory tFactory = (TreeFactory)
```

`FactoryFinder.getFactory(FactoryFinder.TREE_FACTORY);`

1. The default ViewHandler implementation performs a `RequestDispatcher.forward()` call to the web application resource that will actually perform the rendering, so it expects the tree identifier to be the context-relative path (starting with a '/' character) of the web application resource

```
Tree tree = tFactory.getTree(facesContext, treeId);
```

The Tree instance returned by the `getTree()` method will minimally contain a single `UIComponent` provided by the JSF implementation, which will encapsulate any implementation-specific component management that is required. Optionally, a JSF implementation's `TreeFactory` may support the automatic population of the returned `Tree` with additional components, perhaps based on some external metadata description.

JSF 2.4.2.2 Configure the Desired RenderKit

The `Tree` instance provided by the `TreeFactory`, as described in the previous subsection, will automatically be configured to utilize the default `javax.faces.render.RenderKit` implementation provided by the JSF implementation, as described in Section JSF 7.1 "RenderKit". This `RenderKit` must support the standard components and `Renderers` described later in this specification, so its using it will maximize the portability of your application.

However, a different `RenderKit` instance provided by your JSF implementation (or as an add-on library) may be utilized instead, if desired. A reference to this `RenderKit` instance from the standard `RenderKitFactory`, and then assigned to the `Tree` instance created previously, as follows:

```
String renderKitId = ... identifier of desired RenderKit ...;
RenderKitFactory rkFactory = (RenderKitFactory)
    FactoryFinder.getFactory(FactoryFinder.RENDER_KIT_FACTORY);
RenderKit renderKit =
    rkFactory.getRenderKit(renderKitId, facesContext);
tree.setRenderKit(renderKit);
```

As described in Chapter 7, changing the `RenderKit` being used changes the set of `Renderers` that will actually perform decoding and encoding activities. Because the components themselves store only a `rendererType` property (a logical identifier of a particular `Renderer`), it is thus very easy to switch between `RenderKits` (perhaps based on particular characteristics of the servlet request, such as which user agent is being used or the preferred `Locale` of the user), as long as they support `Renderers` with the same `rendererType`s.

JSF 2.4.2.3 Switch To The New Tree

Now that the application has a reference to a `Tree` instance, it can tell the JSF implementation to utilize the new `Tree` as follows:

```
facesContext.setTree(tree);
```

Note that, once this call is performed, the component tree that was previously created by a Faces Request (if any) is no longer accessible. Be sure that you have retrieved any information you want from the previous `Tree` instance before switching.

JSF.2.4.2.4 Configure The Tree's Components

At any time, the application can add new components to the component tree, remove them, or modify the attributes and properties of existing components. For example, a new `FooComponent` (an implementation of `UIComponent`) can be added as a child to the root `UIComponent` in the component tree as follows:

```
FooComponent component = ... create a FooComponent instance ...;  
facesContext.getTree().getRoot().addChild(component);
```

JSF.3

User Interface Component Model

A JSF *user interface component* is the basic building block for creating a JSF user interface. A particular component represents a configurable and reusable element in the user interface, which may range in complexity from simple (such as a button or text field) to compound (such as a tree control or table). Components can optionally be associated with corresponding objects in the data model of application, via *model reference expressions*.

JSF also supports user interface components with several additional support APIs:

- *Converters* -- Pluggable support class to convert the local value of a component to and from the corresponding type in the model tier.
- *Events and Listeners* -- An event broadcast and listener registration model based on the design patterns of the JavaBeans Specification, version 1.0.1.
- *Validators* -- Pluggable support classes that can examine the local value of a component (as received in an incoming request) and ensure that it conforms to the business rules enforced by each Validator. Error messages for validation failures can be generated and sent back to the user during rendering.

The user interface for a particular page of a JSF-based web application is created by assembling the user interface components for a particular request or response into a *component tree*. The components in the tree have parent-child relationships with other components, starting at the *root element* of the tree. Components in the tree can be anonymous or they can be given a *component identifier* by the framework user. Components in the tree can be located based on *component identifiers*, which must be

unique within the scope of the nearest ancestor to the component that is a *naming container*. For complex rendering scenarios, components can also be attached to other components as *facets*.

This chapter describes the basic architecture and APIs for user interface components and the supporting APIs.

JSF.3.1 UIComponent and UIComponentBase

The base interface for all user interface components is

`javax.faces.component.UIComponent`. This interface defines the state information and behavioral contracts for all components through a Java API, which means that components are independent of a rendering technology such as JavaServer Pages (JSP). A standard set of components (described in Chapter 4) that add specialized properties, attributes, and behavior, is also provided as a set of concrete subclasses.

Component writers, tool providers, application developers, and JSF implementors can also create additional `UIComponent` implementations for use within a particular application. To assist such developers, a convenience subclass, `javax.faces.component.UIComponentBase`, is provided as part of JSF. This class provides useful default implementations of nearly every `UIComponent` method, allowing the component writer to focus on the unique characteristics of a particular `UIComponent` implementation.

The following subsections define the key functional capabilities of JSF user interface components.

JSF.3.1.1 Component Type

- `public abstract String getComponentType();`

All concrete `UIComponent` subclasses must override this abstract method and return a non-null value. The component type is used to categorize components by a means other than its Java class name or class parentage. When component rendering is delegated to an external RenderKit, the component type may be used to select one of the available Renderers that know how to visualize components of a particular type.

JSF.3.1.2 Component Identifiers

- `public String getComponentId();`
- `public void setComponentId(String componentId);`

Every component may be named by a *component identifier*, which (if utilized) must be unique among the components that share a common *naming container* parent in a component tree. For maximum portability, component identifiers must conform to the following rules:

- Composed of letters ('a'..'z', 'A'..'Z'), digits ('0'..'9'), dashes ('-'), and underscores ('_') from the USASCII character set.
- Must start with a letter.

To minimize the size of responses generated by JavaServer Faces, it is recommended that component names be as short as possible.

If a component has been given a name, it must be unique in the namespace of the closest ancestor to that component that is a naming container. The root component in the tree must be a naming container.

JSF 3.1.3 Client Identifiers

Client identifiers are used by JSF implementations, as they decode and encode components, for any occasion when the component must have a client side name. Some examples of such an occasion are:

- to name request parameters for a subsequent request from the JSF-generated page.
- to serve as anchors for client side scripting code
- to serve as anchors for client side accessibility labels

The client identifier is derived from the component identifier, if present, or it is generated by the closest ancestor to the component that is a naming container.

- `public String getClientId(FacesContext context);`

For component classes derived from `UIComponentBase`, the client identifier is obtained by executing the following algorithm:

Look up this component's attribute under the name represented by the String constant `UIComponent.CLIENT_ID_ATTR`. If non-null, return it. If null, see if we have a Renderer and if so, delegate to it. If we don't have a Renderer, get the component id for this `UIComponent`. If null, generate one using the closest naming container that is an ancestor of this `UIComponent`, then set the generated id as the component id of this `UIComponent`. Prepend to the component id the component ids of each naming container up to, but not including, the root, separated by the `UIComponent.SEPARATOR_CHAR`. In all cases, save the result as the value of the `UIComponent.CLIENT_ID_ATTR` attribute.

JSF 3.1.4 Component Tree Manipulation

- `public UIComponent getParent();`

Components that have been added as children of another component can identify the parent by calling `getParent()`. For the root node component of a component tree, or any component that is not part of a component tree, this method will return null.

- `public void addChild(UIComponent component);`
- `public void addChild(int index, UIComponent component);`

Adds a new child component to the set of children associated with the current component, either at the end of the list, or at the specified (zero-relative) position.

- `public void clearChildren();`

Remove all children from the child list of the current component.

- `public boolean containsChild(UIComponent component);`

Return `true` if the specified component instance is a child of the current component.

- `public UIComponent getChild(int index);`

Return the component at the specified (zero-relative) position in the child list for this component.

- `public int getChildCount();`

Return the number of child components that are associated with this component.

- `public Iterator getChildren();`

Return an `Iterator` over the child components associated with this component.

- `public Iterator getFacetsAndChildren();`

Return an `Iterator` over the both the facets and child components associated with this component. The all of the facets, if any, are encountered first, in an undetermined order, before the children are encountered, in order.

- `public void removeChild(int index);`
- `public void removeChild(UIComponent component);`

Remove the specified component from the child list for this component.

JSF 3.1.5 Component Tree Navigation

- `public UIComponent findComponent(String expression);`

Find the named component by locating the closest ancestor to this component that is a naming container and calling its `findComponentInNamespace()` method.

JSF.3.1.6 Facet Management

JavaServer Faces supports the traditional model of composing complex components out of simple components via parent-child relationships that organize the entire set of components into a tree, as described in Section JSF.3.1.4 “Component Tree Manipulation”. However, an additional useful facility would be to define particular subordinate components that have a specific *role* with respect to the owning component, which is typically independent of the parent-child relationship. An example might be a “data grid” control, where the children represent the columns to be rendered in the grid. It is useful to be able to identify a component that represents the column header and/or footer, separate from the usual child collection that represents the column data.

To meet this requirement, JavaServer Faces components offer support for *facets*, which represent a named collection of subordinate (but non-child) components that are related to the current component by virtue of a unique *facet name* that represents the role that particular component plays. Although facets are not part of the parent-child tree, they participate in request processing lifecycle methods, as described in Section JSF.3.1.10 “Lifecycle Management Methods”.

- `public void addFacet(String name, UIComponent facet);`
- `public UIComponent getFacet(String name);`
- `public Iterator getFacetNames();`
- `public void removeFacet(String name);`

See the API Javadocs in Section `FIXME` for more details about the functionality of these methods.

For easy use of components that use facets, it is recommended that component authors include type-safe getter and setter methods that correspond to each named facet that is supported by that component class. For example, a component that supports a `header` facet that must be of type `UIHeader` should have methods with signatures and functionality as follows:

```
public UIHeader getHeader() {  
    return ((UIHeader) getFacet("header"));  
}  
  
public void setHeader(UIHeader header) {  
    addFacet("header", header);  
}
```

JSF.3.1.7 Generic Attributes

- `public Object getAttribute(String name);`

- `public Iterator getAttributeNames();`
- `public void setAttribute(String name, Object value);`

The render-independent characteristics of components are generally represented as JavaBean properties with getter and setter methods (see the following section for details). In addition, components may also be associated with generic attributes that are defined outside the component implementation class. Typical uses of generic attributes include:

- Specification of render-dependent characteristics, for use by specific Renderers.
- Configuration parameters to be utilized by Converters, event listeners, and Validators associated with the component.
- General purpose association of application-specific objects with components.

The attributes for a component may be of any Java object type, and are keyed by attribute name (a String). Null attribute values are not allowed; calling `setAttribute()` with a null value parameter is equivalent to removing this attribute. The names of all attributes that currently have a (non-null) value are available via the `getAttributeNames()` method.

Attribute names that begin with “javax.” or “javax.faces” are reserved for use by the JSF implementation.

JSF 3.1.8 Render-Independent Properties

The render-independent characteristics of a user interface component are represented as JavaBean properties, where the method names of the getter and/or setter methods are determined using standard JavaBeans introspection rules, as defined by the JDK class `java.beans.Introspector`. The render-independent properties supported by all `UIComponents` are described in the following table:

Name	Access	Type	Description
<code>componentId</code>	RW	String	The component identifier, as described in Section FIXME/3.1.2 .
<code>componentType</code>	RO	String	The component type, as described in Section FIXME/3.1.1 .
<code>converter</code>	RW	String	The converter identifier of the Converter instance to use when getting and setting the model for this component.
<code>modelReference</code>	RW	String	Model reference expression used to retrieve data values from model tier beans (during <i>Render Response</i> phase), or to transfer component values to model tier beans (during <i>Apply Request Values</i> phase). The expression syntax corresponds to a variable reference in the expression language defined by JSTL 1.0, but without the “\${“ and “}” delimiters.

Name	Access	Type	Description
parent	RO	UIComponent	The parent component for which this component is a child.
rendered	RW	boolean	A flag that, if set to “true”, indicates that this component should be rendered during <i>Render Response</i> phase of the request processing lifecycle.
rendererType	RW	String	Identifier of the Renderer instance (from the set of Renderers supported by the RenderKit associated with the component tree we are processing. If this property is set, several operations during the request processing lifecycle (such as decode() and the encodeXxx() family of methods) will be delegated to a Renderer instance of this type. If this property is not set, the component must implement these methods directly.
rendersChildren	RO	boolean	A flag that, if set to true, indicates that this component manages the rendering of all of its children components (so the JSF implementation should not attempt to render them). The default implementation in UIComponentBase returns false for this property.
rendersSelf	RO	boolean	A flag that, if set to true, indicates that this component has concrete implementations of the decode() and encodeXxx() methods, and is therefore suitable for use even if rendererType is not set. The default implementation in UIComponentBase returns false for this property.
valid	RW	boolean	A flag that, if set to true, indicates that the local value of this component was successfully converted, passes all validations, and was successfully stored during the <i>Update Model Values</i> phase of the request processing lifecycle. JSF classes set this property to false at various points in the request processing lifecycle, as described in Section JSF.2.2 “Standard Request Processing Lifecycle Phases”, to indicate that a problem was encountered.
value	RW	Object	<p>The local value of a JSF component. During the <i>Apply Request Values</i> phase of the request processing lifecycle, the decode() method of the component (or renderer) will convert incoming request parameters into an object representing the new value of the component, and store it in this property.</p> <p>See below for information on the <code>currentValue()</code> method.</p>

The method names for the render-independent property getters and setters must conform to the design patterns in the JavaBeans specification. For boolean properties, the “is” form of the method name must be utilized (so the valid property is retrieved by calling `isValid()`).

To facilitate the use of component classes in tools, all of these JavaBean properties available via getter and setter methods must also be available as attributes under the same name as the property. In other words, a model reference expression stored by a call to the `setModelReference()` will be accessible as an attribute by calling `getAttribute("modelReference")`, and vice versa. This implies that property getter and setter method implementations, in component classes, will generally call the corresponding attribute methods internally.

In addition to the `value` property, which stores a component's local value, `UIComponent` also supports a `currentValue()` method that returns the local value if there is one, or uses the value of the `modelReference` property to retrieve a value from a web tier bean.

JSF 3.1.9 Component Specialization Methods

The methods described in this section are called by the JSF implementation during the various phases of the request processing lifecycle, and may be overridden in a concrete subclass to implement specialized behavior for this component.

- `public boolean broadcast(FacesEvent event, PhaseId phaseId) throws AbortProcessingException;`

The `broadcast()` method is called during the common event processing (see Section JSF 2.3 “Common Event Processing”) at the end of several request processing lifecycle phases. For more information about the event and listener model, see Section JSF 3.4 “Event and Listener Model”.

- `public void decode(FacesContext context) throws IOException;`

This method is called during the *Apply Request Values* phase of the request processing lifecycle, and has the responsibility of extracting a new local value for this component from the request parameters of an incoming request. The default implementation in `UIComponentBase` delegates to a corresponding `Renderer`, if the `rendererType` property is set, and does nothing otherwise.

Generally, component writers will choose to delegate decoding and encoding to a corresponding `Renderer` by setting the `rendererType` property (which means the default behavior described above is adequate). However, overriding this method allows a component writer to create a custom component, complete with its own decoding, encoding, and validation behavior, in a single class if desired.

- `public void encodeBegin(FacesContext context) throws IOException;`
- `public void encodeChildren(FacesContext context) throws IOException;`
- `public void encodeEnd(FacesContext context) throws IOException;`

These methods are called during the *Render Response* phase of the request processing lifecycle, and have the responsibility of creating the response data for the beginning of this component, this component's children (only called if the `rendersChildren` property of this component is `true`), and the ending of this component, respectively. Typically, this will involve generating markup for the output technology being supported, such as creating an HTML `<input>` element for a `UIInput` component. For clients that support it, the encode methods might also generate client-side scripting code (such as JavaScript), and/or stylesheets (such as CSS). The default implementations in `UIComponentBase` delegate to a corresponding `Renderer`, if the `rendererType` property is `true`, and do nothing otherwise.

Generally, component writers will choose to delegate encoding to a corresponding `Renderer`, by setting the `rendererType` property (which means the default behavior described above is adequate). However, overriding these methods allows a component writer to create a custom component, complete with its own decoding, encoding, and validation behavior, in a single class if desired.

- `public void validate(FacesContext context);`

This method is called during the *Process Validations* phase of the request processing lifecycle, and allows the component author to embed any desired correctness checks directly in the component class. The default implementation in `UIComponentBase` performs no correctness checks. For more information about the validation model that JSF provides, see Section JSF.3.5 “Validation Model”.

- `public void updateModel(FaceContext context);`

This method is called during the *Update Model Values* phase of the request processing lifecycle, and is responsible for transferring the local value of this component back to a model tier bean via the model reference expression (with appropriate conversions as necessary). The default implementation in `UIComponentBase` calls the `setModelValue()` method on the `FacesContext` instance for the current request, and then clears the local value, as described in the JavaDocs for this method.

JSF.3.1.10 Lifecycle Management Methods

The following methods are called by the various phases of the request processing lifecycle, and implement a recursive tree walk of the components in a component tree, calling the component specialization methods described above for each component. These methods are not generally overridden by component writers, but doing so may be useful for some advanced component implementations.

- `public void processDecodes(FacesContext context) throws IOException;`

This method is called (on the root component of a component tree) during the *Apply Request Values* phase of the request processing lifecycle. The default implementation in `UIComponentBase` ensures that the `processDecodes()` method of all facets of this component, then all children of this component, followed by the `decode()` method of this component, are called.

- `public void processValidators(FacesContext context);`

This method is called (on the root component of a component tree) during the *Process Validations* phase of the request processing lifecycle. The default implementation in `UIComponentBase` ensures that the `processValidators()` method of all facets and children of this component are called appropriately, followed by calls to the `validate()` method of all validators registered to this component, and then the `validate()` method of this component itself.

- `public void processUpdates(FacesContext context);`

This method is called (on the root component of a component tree) during the *Update Model Values* phase of the request processing lifecycle. The default implementation in `UIComponentBase` ensures that the `processUpdates()` method of all facets of this component, then all children of this component, followed by the `updateModel()` method of this component, are called.

JSF.3.2 NamingContainer and UINamingContainer

The `NamingContainer` interface defines a simple, flexible way for components to have names on the client and server side of a Faces application. The API provides a default implementation of `NamingContainer` in the class `UINamingContainer`. Any implementation of `NamingContainer` must also implement `UIComponent`.

Any JSF implementation must make the root of the component tree implement `NamingContainer`. This is most easily accomplished by having the root be an instance of `UINamingContainer`.

- `public void addComponentToNamespace(UIComponent namedComponent);`
- `public void removeComponentFromNamespace(UIComponent namedComponent);`
- `public UIComponent findComponentInNamespace(String name);`
- `public String generateClientId();`

The add and remove methods above simply add to or remove from the namespace of this naming container, using the *component identifier* of the argument component as the name.

`findComponentInNamespace()` looks in the namespace of this naming container for a component under the argument name. If the argument name contains `UIComponent.SEPARATOR_CHAR` characters, each segment between the `UIComponent.SEPARATOR_CHAR` characters is treated as a component identifier in the previous naming container's namespace.

Consider the following usage:

```
namingComponent.findComponent( "containerOne.containerTwo.leafTwo" ), where namingComponent is a UIComponent that implements NamingContainer. This call will end up calling NamingContainer.findComponentInNamespace(). The first two segments must be the component identifiers of NamingContainer instances. containerOne is in namingComponent's namespace, containerTwo is in containerOne's namespace, and leafTwo is in containerTwo's namespace.
```

`generateClientId()` returns a string of the form “`idN`” where `N` is a serial number generated by the `NamingContainer`. This method is called when the *client identifier* of a component is requested, but the component has no component identifier.

JSF.3.3 Conversion Model

JSF.3.3.1 Overview

A typical web application must constantly deal with two fundamentally different viewpoints of the underlying data being manipulated through the user interface:

- The *model* view -- Data is typically represented as Java objects (often JavaBeans), with data represented in some native Java datatype. For example, date and time values might be represented in the model view as instances of `java.util.Date` or `java.util.Calendar`.
- The *presentation* view -- Data is typically represented in some form that can be perceived or modified by the user of the application. For example, a date or type value might be represented as a text string, as three text strings (one each for month/date/year or one each for hour/minute/second), as a calendar control, associated with a spin control that lets you increment or decrement individual elements of the date or time with a single mouse click, or in a variety of other ways. Some presentation views may depend on the preferred language or locale of the user (such as the commonly used `mm/dd/yy` and `dd/mm/yy` date formats, or the variety of punctuation characters in monetary amount presentations for various currencies).

To transform data formats between these views, JavaServer Faces provides an ability to plug in an optional `Converter` for each `UIComponent`, which has the responsibility of converting the internal data representation between the two views. The application developer attaches a particular `Converter` to a particular

`UIComponent` by calling `setConverter()`, passing the `converterId` of the particular converter. There are two mechanisms by which a `Converter` can be supplied via this attribute:

- You can supply an instance of a class that implements the `Converter` interface. This interface is described in Section JSF 3.3.2 “Converter”. You can supply a String whose value is the *converter identifier* of a `Converter` implementation that is acquired from the `ConverterFactory` provided by to your application by the JSF implementation, or indirectly from your application having registered a `Converter` instance with the `ConverterFactory` provided by the JSF implementation. `ConverterFactory` is described in Section JSF 3.3.3 “ConverterFactory”.

JSF 3.3.2 Converter

JSF provides the `javax.faces.convert.Converter` interface to define the behavioral characteristics of a `Converter`. Instances of implementations of this interface are identified by a *converter identifier*, which can be registered with, and later retrieved from, a `ConverterFactory`, as described in Section JSF 3.3.3 “ConverterFactory”.

Typically, a `Converter` will be a stateless object that requires no extra configuration information to perform its responsibilities. However, in some cases, it is useful to provide configuration parameters to the `Converter` (such as a `java.text.DateFormat` pattern for a `Converter` that supports `java.util.Date` model objects). Such configuration information may be provided via named attributes of the `UIComponent` instance that is passed to the `Converter`’s methods. The attribute names used by a `Converter` to configure its behavior should be documented in the usage instructions for that `Converter`, and should be selected to not conflict with other render-independent and render-dependent attributes of the underlying component.

`Converter` implementations should be programmed so that the conversions they perform are symmetric. In other words, if a model data object is converted to a `String` (via a call to the `getAsString()` method), it should be possible to call `getAsObject()` and pass it the converted `String` as the `value` parameter, and return a model data object that is semantically equal to the original one.

The following two method signatures are defined by the `Converter` interface:

- `public Object getAsObject(FacesContext context, UIComponent component, String value) throws ConverterException;`

This method is used to convert the presentation view of a component’s value (typically a `String` that was received as a request parameter) into the corresponding model view. It is called during the *Apply Request Values* phase of the request processing lifecycle.

```
■ public String getAsString(FacesContext context, UIComponent component, Object value) throws ConverterException;
```

This method is used to convert the model view of a component's value (typically some native Java class) into the presentation view (typically a String that will be rendered in some markup language). It is called during the *Render Response* phase of the request processing lifecycle.

JSF.3.3.3 ConverterFactory

A single instance of `javax.faces.convertConverterFactory` must be made available to each JSF-based web application. The factory instance can be acquired by JSF implementations, or by application code, as follows:

```
ConverterFactory factory = (ConverterFactory)
    FactoryFinder.getFactory(FactoryFinder.CONVERTER_FACTORY);
```

The `ConverterFactory` instance that is returned will support the following functionality:

```
■ public Converter getConverter(String converterId);
```

Create (if necessary) and return a `Converter` instance that has been previously registered (via a call to `addConverter()`, described below) for the specified converter identifier. Using converter identifiers allows page authors and application developers to be insulated from knowing the actual `Converter` implementation class that should be used.

```
■ public Iterator getConverterIds();
```

Return an `Iterator` over the converter identifiers known to this `ConverterFactory`.

```
■ public void addConverter(String converterId, Converter converter);
```

At any time, a new `Converter` implementation can be registered, associated with a new converter identifier. Typically, JSF implementations, application frameworks, and/or application components will register the `Converters` that they support at application initialization time.

JSF.3.4 Standard Converter Implementations

Every JSF implementation must provide a set of standard `Converter` implementations, as described in the table on page 47, preregistered with the `ConverterFactory` instance that is accessible as described in Section JSF.3.3.3 “ConverterFactory”. Each of the `Converters` must implement `Object-to-String` and `String-to-Object` conversions for the `Object` types listed in the *Model DataType* column. In addition, the behavior of each of these `Converters` is customized by

adding *configuration attributes* to the `UIComponent` instance that will be passed to the `getAsObject()` and `getAsString()` methods. The meaning, valid values, and default value (if no such attribute value is specified on the component) for these configuration attributes are as follows:

- *dateStyle* - Date style pattern name as defined by the `java.text.DateFormat` class. Valid values: `short`, `medium`, `long`, or `full`. Default value: `short`.
- *formatPattern* - String version of a formatting pattern suitable for use with a `java.text.DateFormat` or `java.text.NumberFormat` instance. There is no default for this attribute; attempting to use a `Converter` requiring this attribute, but without specifying a value, must result in a `ConverterException`.
- *numberStyle* - Number style pattern name as defined by the `java.text.NumberFormat` class. Valid values: `currency`, `integer`, `number`, or `percent`. Default value: `integer`.
- *timeStyle* - Time style pattern name as defined by the `java.text.DateFormat` class. Valid values: `short`, `medium`, `long`, or `full`. Default value: `short`.
- *timezone* - A time zone identifier as defined by the `java.util.TimeZone` class. If not specified, no time zone adjustment is performed on the constructed `DateFormat`.

JSF.3.4 Event and Listener Model

JSF.3.4.1 Overview

JSF implements a model for event notification and listener registration based on the design patterns in the JavaBeans Specification, version 1.0.1. This is similar to the approach taken in other user interface toolkits, such as the Swing Framework included in the JDK.

A `UIComponent` subclass may choose to emit *events* that signify significant state changes, and broadcast them to *listeners* that have registered an interest in receiving events of the type indicated by the event's implementation class. At the end of several phases of the request processing lifecycle, the JSF implementation will broadcast all of the events that have been queued to interested listeners.

JSF.3.4.2 Event Classes

All events that are broadcast by JSF user interface components must extend the `javax.faces.event.FacesEvent` base class. The parameter list for the constructor(s) of this event class must include a `UIComponent`, which identifies the component from which the event will be broadcast to interested listeners. The source

TABLE 3-1 Standard Converter Implementations

Converter Id	Model Datatype	Configuration Attributes	Description
Date	Date, Number ¹	dateStyle, timezone	Create a localized DateFormat date instance for the specified dateStyle, set the time zone (if specified), and use its format() and parse() methods to convert.
DateFormat	Date, Number	formatPattern, timezone	Create a localized SimpleDateFormat instance based on the specified formatPattern, set the time zone (if specified), and use its format() and parse() methods to convert.
DateTime	Date, Number	dateStyle, timeStyle, timezone	Create a localized DateFormat date/time instance for the specified dateStyle and timeStyle, set time zone (if specified), and use its format() and parse() methods to convert.
Number	Number	numberStyle	Create a localized NumberFormat instance based on the specified numberStyle, and use its format() and parse() methods to convert.
NumberFormat	Number	formatPattern	Create a localized NumberFormat instance based on the specified formatPattern, and use its format() and parse() methods to convert.

1. If a Converter for a date/time value is passed a java.lang.Number, or any of its subclasses (such as java.lang.Long), the longValue() method must be used to extract a value in milliseconds, which can be used as the argument to the java.util.Date constructor to construct the corresponding Date value.

component can be retrieved from the event object itself by calling getComponent(). Additional constructor parameters and/or properties on the event class can be used to communicate additional information about the event.

In conformance to the naming patterns defined in the JavaBeans Specification, event classes typically have a class name that ends with “Event”. It is recommended that application event classes follow this naming pattern as well.

JSF includes two standard FacesEvent subclasses, which are emitted by the corresponding standard UIComponent subclasses described in the following chapter.

- ActionEvent -- Emitted by a UICommand component when the user activates the corresponding user interface control (such as a clicking a button or a hyperlink).
- ValueChangedEvent -- Emitted by a UIInput component (or appropriate subclass) when a new local value has been created, and has passed all validations.

JSF 3.4.3 Listener Classes

For each event type that may be emitted, a corresponding listener interface must be created, which extends the `javax.faces.event.FacesListener` interface. The method signature(s) defined by the listener interface must take a single parameter, an instance of the event class for which this listener is being created. A listener implementation class will implement one or more of these listener interfaces, along with the event handling method(s) specified by those interfaces. The event handling methods will be called during event broadcast, one per event.

In addition to the event handling methods, `FacesListener` defines a `getPhaseId()` method, which must return a phase identifier (of type `PhaseId`, see Section JSF 3.4.4 “Phase Identifiers”) describing the request processing lifecycle phase during which this listener is interested in receiving events.

In conformance to the naming patterns defined in the JavaBeans Specification, listener interfaces have a class name based on the class name of the event being listened to, but with the word “Listener” replacing the trailing “Event” of the event class name (thus, the listener for a `FooEvent` would be a `FooListener`). It is recommended that application event listener interfaces follow this naming pattern as well.

Corresponding to the two standard event classes described in the previous section, JSF defines two standard event listener interfaces that may be implemented by application classes:

- `ActionListener` -- a listener that is interested in receiving `ActionEvent` events, received on its `processAction()` method.
- `ValueChangeListener` -- a listener that is interested in receiving `ValueChangeEvent` events, received on its `processValueChanged()` method.

JSF 3.4.4 Phase Identifiers

As described in Section JSF 2.3 “Common Event Processing”, event handling occurs at the end of several phases of the request processing lifecycle. In addition, a particular listener must indicate, through the value it returns from the `getPhaseId()` method, the phase(s) in which it wishes to be notified. This indication is done by returning an instance of `javax.faces.event.PhaseId`. This class defines a typesafe enumeration of all the legal values that may be returned by `getPhaseId()`. In addition, a special value (`PhaseId.ANY_PHASE`) may be returned to indicate that this listener wants to be notified of events it is registered for, no matter what request processing lifecycle phase is currently being performed.

JSF.3.4.5 Listener Registration

A concrete `UIComponent` subclass that emits events of a particular type must include public methods to register and deregister a listener implementation. In order to be recognized by development tools, these listener methods must follow the naming patterns defined in the JavaBeans Specification. For example, for a component that emits `FooEvent` events, to be received by listeners that implement the `FooListener` interface, the method signatures (on the component class) must be:

- `public void addFooListener(FooListener listener);`
- `public void removeFooListener(FooListener listener);`

The application (or other components) may register listener instances at any time, by calling the appropriate “add” method. The set of listeners associated with a component is part of the state information that JSF saves and restores (so listener implementation classes must be `Serializable`).

The `UICommand` and `UIInput` standard component classes include listener registration and deregistration methods for event listeners associated with the event types that they emit. The `UIInput` methods are also inherited by `UIInput` subclasses, including `UISelectBoolean`, `UISelectMany`, and `UISelectOne`.

JSF.3.4.6 Event Queueing

During the processing being performed by any phase of the request processing lifecycle, events may be created and queued by calling the `addFacesEvent()` method on the `FacesContext` instance for the current request. At the end of the phase, any queued events will be broadcast to interested listeners in the order that the events were originally queued.

Deferring event broadcast until the end of a request processing lifecycle phase ensures that the entire component tree has been processed by that state, and that event listeners all see the same consistent state of the entire tree, no matter when the event was actually queued.

JSF.3.4.7 Event Broadcasting

As described in Section JSF.2.3 “Common Event Processing”, at the end of each request processing lifecycle phase that may cause events to be queued, the JSF implementation will iterate over the queued events and call the `broadcast()` method to actually notify the registered listeners. The `broadcast()` method’s responsibilities include:

- Based on the implementation class of the `FacesEvent` that is passed as a parameter, identify the list of listeners who have registered an interest in that event type (that is, if a `FooEvent` is passed, the `broadcast()` method must identify listeners registered on this component by calls to `addFooListener()`).
- For each interested listener, call the appropriate event processing method on that listener instance, passing the event that has occurred (cast to the appropriate concrete event type class).
- Determine whether there are any further listeners interested in this event (registered with an interest in a future phase, or registered for event processing in any phase). If there are any further interested listeners, return `true`; otherwise, return `false`.

During event broadcasting, a listener processing an event may:

- Examine or modify the state of any component in the component tree.
- Add or remove components from the component tree.
- Add messages to be returned to the user, by calling `addMessage()` on the `FacesContext` instance for the current request.
- Queue one or more additional events, from the same source component or a different one, for processing during the current lifecycle phase.
- Throw an `AbortProcessingException`, to tell the JSF implementation that no further broadcast of this event should take place.
- Call `renderResponse()` on the `FacesContext` instance for the current request. This tells the JSF implementation that, when the current phase of the request processing lifecycle has been completed, control should be transferred to the *Render Response* phase.
- Call `responseComplete()` on the `FacesContext` instance for the current request. This tells the JSF implementation that, when the current phase of the request processing lifecycle has been completed, processing for this request should be terminated (because the actual response content has been generated by some other means).

JSF 3.4.8 Application Events

[ED. NOTE -- This information in this section will change in the next public version of this specification. The following describes the current placeholder implementation for communicating form events to an application handler.]

All of the event processing described earlier in this section has related to *user interface* events, whose primary purpose is to modify (in some way) the state of the component tree and possibly causing the component tree to be rerendered. An additional mechanism is necessary in order to trigger higher level application business logic components, typically based on values from more than one

component in the tree. In the typical HTML-based web browser scenario, for example, a form submit is an example of this use case. Such a case can be handled by extending the event processing model to include the idea of *application events*.

JSF provides an abstract base class for application events, `javax.faces.event.ApplicationEvent`, and a concrete subclass for form submit events, `FormEvent`. Form events may be queued to the application by calling the `addApplicationEvent()` method on the `FacesContext` instance for the current request. A standard `UICommand` component representing a submit button queues such an event, in addition to the `ActionEvent` described above, when appropriate.

Queued application events will be processed during the *Invoke Application* phase of the request processing lifecycle, as described in Section JSF.2.2.5 “Invoke Application”. Each queued event will be transmitted to the `ApplicationHandler` instance for this application, as described in Section JSF.6.3 “ApplicationHandler”.

JSF.3.5 Validation Model

JSF.3.5.1 Overview

JSF supports a mechanism for registering zero or more *validators* on each component in the component tree. A validator’s purpose is to perform correctness checks on the local value of the component, during the *Process Validations* phase of the request processing lifecycle. In addition, a component may implement internal correctness checking in a `validate()` method that is part of the component class.

JSF.3.5.2 Validator Classes

A validator must implement the `javax.faces.validator.Validator` interface, which contains a `validate()` method signature. A convenience base class (`ValidatorBase`) is also provided.. It is recommended that application-provided validators subclass `ValidatorBase`, rather than implement `Validator`, to provide protection against future evolution of the `Validator` interface.

General purpose validators may require configuration values in order to define the precise correctness check to be performed. For example, a validator that enforces a maximum length might wish to support a configurable length limit. Such configuration values are typically implemented as JavaBeans properties, and/or constructor arguments, on the Validator implementation class. In addition, a validator may elect to use generic attributes of the component being validated for configuration information.

JSF includes implementations of several standard validators, as described in Section JSF.3.5.5 “Standard Validator Implementations”.

JSF 3.5.3 Validation Registration

The `UIComponent` interface includes an `addValidator()` method to register an additional validator for this component, and a `removeValidator()` method to remove an existing registration.

The application (or other components) may register validator instances at any time, by calling the `addValidator()` method. The set of validators associated with a component is part of the state information that JSF saves and restores (so validator implementation classes must be `Serializable`).

JSF 3.5.4 Validation Processing

During the *Process Validations* phase of the request processing lifecycle (as described in Section JSF 2.2.3 “Process Validations”), the JSF implementation will ensure that the `validate()` method of each registered `Validator`, as well as the `validate()` method of the component itself, is called for each component in the component tree. The responsibilities of each `validate()` method include:

- Perform the correctness check for which this validator was registered.
- If violation(s) of the correctness rules are found, set the `valid` property of this component to `false`, and add zero or more messages describing the problem by calling `addMessage()` on the `FacesContext` instance for the current request.

In addition, a `validate()` method may:

- Examine or modify the state of any component in the component tree.
- Add or remove components from the component tree.
- Queue one or more events, from the same component or a different one, for processing during the current lifecycle phase.
- Call `renderResponse()` on the `FacesContext` instance for the current request. This tells the JSF implementation that, when the current phase of the request processing lifecycle has been completed, control should be transferred to the *Render Response* phase.
- Call `responseComplete()` on the `FacesContext` instance for the current request. This tells the JSF implementation that, when the current phase of the request processing lifecycle has been completed, processing for this request should be terminated (because the actual response content has been generated by some other means).

JSF 3.5.5 Standard Validator Implementations

JavaServer Faces defines a standard suite of `Validator` implementations that perform a variety of commonly required correctness checks. In addition, component writers, application developers, and tool providers will often define additional

Validator implementations that may be used to support component-type-specific or application-specific constraints. These implementations share the following common characteristics:

- Standard Validators accept configuration information as either parameters to the constructor that creates a new instance of that Validator, or as JavaBeans properties on the Validator implementation class.
- To support internationalization, Message instances should be created by passing a *message identifier* (along with optional substitution parameters) to an appropriate MessageResources instance. The message identifiers for such standard messages are also defined by manifest String constants in the implementation classes.
- Unless otherwise specified, components with a null local value cause the validation checking by this Validator to be skipped. If a component should be required to have a non-null value, a separate instance of RequiredValidator should be registered in order to enforce this rule.

Localizable message template strings may contain placeholder elements, as defined in the API JavaDocs for the `java.text.MessageFormat` class. Several of the `getMessage()` method signatures on the `MessageResources` class allow replacement value(s) for these placeholders to be included. The definitions of the standard message identifiers, below, will note cases where replacement value(s) will be included in the message addition.

The following standard Validator implementations (in package `javax.faces.validator`) are provided:

- `DoubleRangeValidator` -- Checks the local value of a component, which must be of any numeric type (or whose String value is convertible to `double`), against specified maximum and/or minimum values.
- `LengthValidator` -- Checks the length (i.e. number of characters) of the local value of a component, which must be of type `String` (or convertible to a `String`), against maximum and/or minimum values.
- `LongRangeValidator` -- Checks the local value of a component, which must be of any numeric type convertible to `long` (or whose String value is convertible to `long`), against maximum and/or minimum values.
- `RequiredValidator` -- Checks the local value of a component for being non-null. If the local value is a String, it must also have a length of at least one character.
- `StringRangeValidator` -- Checks the local value of a component, which must be of type `String` (or convertible to a `String`), against specified maximum and/or minimum values.

JSF.4

Standard User Interface Components

In addition to the interface `UIComponent` and the abstract base class `UIComponentBase`, described in the previous chapter, JSF provides a number of concrete user interface component implementation classes that cover the most common requirements. In addition, component writers will typically create new components by subclassing one of the standard component classes (or the `UIComponentBase` class). It is anticipated that the number of standard component classes will grow in future versions of the JavaServer Faces specification. All of the concrete classes are part of the `javax.faces.component` package, and (unless otherwise noted) are subclasses of `javax.faces.component.UIComponentBase`.

Each of these classes defines the render-independent characteristics of the corresponding component as JavaBeans properties. The class descriptions also specify minimal implementations of the `decode()` (where appropriate) and `encodeXXX()` methods, which implement very basic decoding and encoding that is compatible with HTML/4.01. It is assumed, however, that rendering will normally be delegated to `Renderer` instances acquired from the `RenderKit` associated with our component tree.

The standard `UIComponent` subclasses also define a symbolic String constant named `TYPE`, which specifies the value that will be returned by `getComponentType()` for this class, which will be equal to the fully qualified class name of the standard component implementation class.

Component types should be globally unique across JSF implementations. The suggested convention is to use a “package name” syntax similar to that used for Java packages, although the types do not need to correspond to class names. Component type values starting with “javax.faces.*” are reserved for use by standard component types defined in this Specification.

JSF.4.1 UICommand

The `UICommand` class represents a user interface component which, when activated by the user, triggers an application-specific “command” or “action”. Such a component is typically rendered as a push button, a menu item, or a hyperlink.

- `public String getCommandName();`
- `public void setCommandName(String commandName);`

The `commandName` property getter and setter methods are typesafe aliases for `getValue()` and `setValue()`. This allows the command name to be retrieved (via a call to `currentValue()`) from the locally configured value, or indirectly via the model reference expression.

The `commandName` property identifies the command or action that should be executed when this command is activated. Command names need not be unique across a single component tree; it is common to provide users more than one way to request the same command. They must be comprised of characters that are legal in a URL.

The default rendering functionality for `UICommand` components is:

- `decode()` - If there is a request parameter on the incoming request that matches the client identifier of this component, enqueue an `ActionEvent` to be broadcast to registered event listeners, as described in Section JSF.3.4 “Event and Listener Model”. For backwards compatibility with the current application event handling model (which will be revised in the next public draft of this Specification), also enqueue a `FormEvent` to the application, passing the form name of the form being submitted and the command name of this `UICommand`.
- `encodeBegin()` - Render the beginning of an HTML submit button, using the client identifier of this component as the “name” attribute, and “submit” as the value to be returned to the server if this button is selected by the user.
- `encodeEnd()` - Render the ending of an HTML submit button.

JSF.4.2 UIForm

The `UIForm` class (extends `UIOutput`) represents a user interface component that corresponds to an input form to be presented to the user, and whose child components represent (among other things) the input fields to be included when the form is submitted.

- `public String getFormName();`

```
■ public void setFormName(String formName);
```

The `formName` property getter and setter methods are typesafe aliases for `getValue()` and `setValue()`. This allows the form name to be retrieved (via a call to `currentValue()`) from the locally configured value, or indirectly via the model reference expression.

The `formName` property identifies the form that is being submitted, and will generally be used by the application (during the *Invoke Application* phase of the request processing lifecycle) to dispatch to the corresponding application logic for processing this form.. They must be comprised of characters that are legal in a URL.

The default rendering functionality for `UIForm` components is:

- `encodeBegin()` - Render an HTML `<form>` element, with an `action` attribute calculated as “`/xxxxx/faces/yyyyy`”, where “`xxxxx`” is the context path of the current web application, “`yyyyy`” is the response tree identifier of the page being rendered. URL rewriting will have been applied, to maintain session identity in the absence of cookies.
- `encodeEnd()` - Render the `</form>` element required to close the `<form>` element created during `encodeBegin()`.

JSF.4.3 `UIGraphic`

The `UIGraphic` class (extends `UIOutput`) represents a user interface component that displays a graphical image to the user. The user cannot manipulate this component; it is for display purposes only.

```
■ public String getURL();  
■ public void setURL(String url);
```

The `url` property getter and setter methods are typesafe aliases for `getValue()` and `setValue()`. This allows the URL of the image to be retrieved (via a call to `currentValue()`) from the locally configured value, or indirectly via the model reference expression.

The `url` property contains a URL for the image to be displayed by this component. If the value begins with a slash ('/') character, it is assumed to be relative to the context path of the current web application. Otherwise, the URL is used without modification.

The default rendering functionality for `UIGraphic` components is:

- `encodeEnd()` - Render an HTML `` element with a `src` element based on the local value of this component. URL rewriting will have been applied, to maintain session identity in the absence of cookies.

JSF.4.4 UIInput

The `UIInput` class represents a user interface component that both displays the current value of the component to the user (as `UIOutput` components do), and also causes the inclusion of updated values on the subsequent request, which will be decoded during the *Apply Request Values* phase of the request processing lifecycle. There are no restrictions on the data type of the local value, or the object referenced by the model reference expression (if any); however, individual `Renderers` will generally impose restrictions on the types of data they know how to display.

The default rendering functionality for `UIInput` components is:

- `decode()` - Set the local value to the value of the corresponding request parameter, or null if the corresponding request parameter is not present.
- `encode()` - Retrieve the current value of this component, convert it to a String (if necessary), and render an HTML `<input type="text">` element.

`UIInput` is a source of `ValueChangeEvent` events, which are emitted when a new value has been established for an input component, and all validations have been passed successfully. (Subclasses of `UIInput` also inherit this behavior.) It includes methods to register and deregister `ValueChangedListener` instances interested in those events. See Section JSF.3.4 “Event and Listener Model” for more details on the event and listener model provided by JSF.

JSF.4.5 UIOutput

The `UIOutput` class represents a user interface component that displays the unmodified current value of this component to the user. The user cannot manipulate this component; it is for display purposes only. There are no restrictions on the data type of the local value, or the object referenced by the model reference expression (if any); however, individual `Renderers` will generally impose restrictions on the types of data they know how to display.

The default rendering functionality for `UIOutput` components is:

- `encodeEnd()` - Retrieve the current value of this component, convert it to a String (if necessary), and render directly to the response.

Because `UIOutput` components are write only, this class must override the `updateModel()` method inherited from `UIComponent` and perform no model update processing.

JSF.4.6 UIPanel

The `UIPanel` class (extends `UIOutput`) represents a user interface component that is primarily a container for its children. The default implementation sets its `rendersChildren` property to `true`, but has no rendering behavior of its own.

Subclasses of `UIPanel` will typically implement layout management for child components, either directly (via its `encodeXXX()` methods) or indirectly (via implementation in an appropriate Renderer). The latter scenario -- delegating rendering to a Renderer -- means that `UIPanel` can be used as a component implementation class for most layout management purposes, without the need to subclass.

JSF.4.7 UIPparameter

The `UIParameter` class (extends `UIOutput`) represents a user interface component that has no visible rendering behavior. Instead, it provides a convenient mechanism to provide configuration parameter values to a parent component (such as substitution parameters for an internationalized message being processed with `java.text.MessageFormat`, or adding request parameters to a generated hyperlink).

Parent components should retrieve the value of a parameter component by calling `currentValue()`. In this way, the parameter value can be set directly on the component (via `setValue()`) or retrieved indirectly via the model reference expression.

- `public String getName();`
- `public void setName(String name);`

In some scenarios, it is necessary to provide a parameter name, in addition to the parameter value that is accessible via the `currentValue()` method. Renderers that support or require parameter names on their nested `UIParameter` child components should document their use of this property.

This component has no default rendering functionality.

JSF.4.8 UISelectBoolean

The `UISelectBoolean` class (extends `UIInput`) represents a user interface component which can have a single, boolean value of `true` or `false`. It is most commonly rendered as a checkbox.

- `public boolean isSelected();`
- `public void setSelected(boolean selected);`

The `selected` property getter and setter methods are typesafe aliases for `getValue()` and `setValue()`. This allows the selected state of the component to be retrieved (via a call to `currentValue()`) from the locally configured value, or indirectly via the model reference expression.

The default rendering functionality for `UISelectBoolean` components is:

- `decode()` - Set the local value to `true` or `false`, based on the corresponding request parameter included with this request (if any).
- `encodeBegin()` - Render an HTML `<input type="checkbox">` element.

As a subclass of `UIInput`, `UISelectBoolean` inherits the capability to broadcast `ValueChangeEvent` events, as described in Section JSF.3.4 “Event and Listener Model”.

JSF.4.9 SelectItem

`SelectItem` is a utility class representing a single choice, from among those made available to the user, for a `UISelectMany` or `UISelectOne` component. It is not itself a `UIComponent` subclass.

- `SelectItem(Object value, String label, String description);`
- `public String getDescription();`
- `public String getLabel();`
- `public Object getValue();`

Each `SelectItem` has three immutable properties: `value` is the value that will be returned (as a request parameter) if the user selects this item, `label` is the visible content that enables the user to determine the meaning of this item, and `description` is a description of this item that may be used within development tools, but is not rendered as part of the response.

JSF.4.10 UISelectItem

The `UISelectItem` class represents a single `SelectItem` that will be included in the list of available options in a `UISelectMany` or `UISelectOne` component that is the direct parent of this component. This component has no decoding or encoding behavior of its own -- its purpose is simply to configure the behavior of the parent component.

- `public String getItemDescription();`
- `public void setItemDescription(String itemDescription);`
- `public String getItemLabel();`
- `public void setItemLabel(String itemLabel);`
- `public String getItemValue();`
- `public void setItemValue(String itemValue);`

These attributes are used to configure an instance of `SelectItem` that will be added to the set of available options for our parent `UISelectMany` or `UISelectOne` tag.

JSF.4.11 UISelectItems

The `UISelectItem` class represents a set of `SelectItem` instances that will be included in the list of available options in a `UISelectMany` or `UISelectOne` component that is the direct parent of this component. This component has no decoding or encoding behavior of its own -- its purpose is simply to configure the behavior of the parent component.

When assembling the list of available options, our parent `UISelectMany` or `UISelectOne` tag must use the `currentValue()` method to acquire the current value of this component. This value must be of one of the following types, which causes the specified behavior:

- Single instance of `SelectItem` -- This instance is added to the set of available options.
- Array of `SelectItem` -- Each instance in this array is added to the set of available options, in ascending order.
- Collection of `SelectItem`¹ -- Each element in this Collection will be added to the set of available options, in the order that an iterator over the Collection returns them.

JSF.4.12 UISelectMany

The `UISelectMany` class (extends `UIInput`) represents a user interface component that represents the user's choice of a zero or more items from among a discrete set of available items. It is typically rendered as a combo box or a set of checkboxes.

- `public Object[] getSelectedValues();`
- `public void setSelectedValues(Object selectedValues[]);`

The `selectedValues` property getter and setter methods are typesafe aliases for `getValue()` and `setValue()`. This allows the values of the currently selected items (if any) to be retrieved (via a call to `currentValue()`) from the locally configured value, or indirectly via the model reference expression.

The `selectedValues` property contains the *values* of the currently selected items (if any). If there is no current value, or none of the specified values match the value of any item on the items list, the component may be rendered with no item currently selected.

The list of available items for this component is specified by nesting zero or more child components of type `UISelectItem` (see Section JSF.4.10 "UISelectItem") or `UISelectItems` (see Section JSF.4.11 "UISelectItems") inside this component.

The default rendering functionality for `UISelectMany` components is:

- `decode()` - Set the local value to a String array containing the values of the corresponding request parameter, or `null` if the corresponding request parameter is not present.

1. This includes any implementation of `java.util.List` and `java.util.Set`.

- `encodeEnd()` - Retrieve the current value of this component, convert it to a String (if necessary), and render an HTML <select> element. As the available items are rendered, any item whose value matches one of the Strings in the current value of this component will cause this item to be preselected.

As a subclass of `UIInput`, `UISelectMany` inherits the capability to broadcast `ValueChangeEvent` events, as described in Section JSF.3.4 “Event and Listener Model”.

JSF.4.13 `UISelectOne`

The `UISelectOne` class (extends `UIInput`) represents a user interface component that represents the user’s choice of a single item from among a discrete set of available items. It is typically rendered as a combo box or a set of radio buttons.

- `public Object getSelectedValue();`
- `public void setSelectedValue(Object selectedValue);`

The `selectedValue` property getter and setter methods are typesafe aliases for `getValue()` and `setValue()`. This allows the value of the currently selected item (if any) to be retrieved (via a call to `currentValue()`) from the locally configured value, or indirectly via the model reference expression.

The `selectedValue` property contains the *value* of the currently selected item (if any). If there is no current value, or the specified value does not match the value of any item on the items list, the component may be rendered with no item currently selected.

The list of available items for this component is specified by nesting zero or more child components of type `UISelectItem` (see Section JSF.4.10 “`UISelectItem`”) or `UISelectItems` (see Section JSF.4.11 “`UISelectItems`”) inside this component.

The default rendering functionality for `UISelectOne` components is:

- `decode()` - Set the local value to the value of the corresponding request parameter, or `null` if the corresponding request parameter is not present.
- `encodeEnd()` - Retrieve the current value of this component, convert it to a String (if necessary), and render an HTML <select> element. As the available items are rendered, any item whose value matches the current value of this component will cause this item to be preselected.

As a subclass of `UIInput`, `UISelectOne` inherits the capability to broadcast `ValueChangeEvent` events, as described in Section JSF.3.4 “Event and Listener Model”.

JSF.5

Per-Request State Information

During request processing for a JSF page, a context object is used to represent request-specific information, as well as provide access to services for the application. This chapter describes the classes which encapsulate this contextual information.

JSF.5.1 FacesContext

JSF defines the `javax.faces.context.FacesContext` abstract base class for representing all of the contextual information associated with a processing an incoming request, and creating the corresponding response. A `FacesContext` instance is created by the JSF implementation, prior to beginning the request processing lifecycle, by a call to the `getFacesContext()` method of `FacesContextFactory`, as described in Section [FIXME](#), below. When the request processing lifecycle has been completed, the JSF implementation will call the `release()` method, which gives JSF implementations the opportunity to release any acquired resources, as well as to pool and recycle `FacesContext` instances rather than creating new ones for each request.

JSF.5.1.1 Servlet API Components

[ED. NOTE -- Direct access to servlet API objects will likely be abstracted away in a future version of this specification.]

- `public HttpSession getHttpSession();`
- `public ServletContext getServletContext();`
- `public ServletRequest getServletRequest();`

- `public ServletResponse getServletResponse();`

The `FacesContext` instance provides immediate access to all of the components defined by the servlet container within which a JSF-based web application is deployed. The `getHttpSession()` method will only return a non-null session instance if the current request is actually an `HttpServletRequest`, and a session is already in existence. To create and acquire a reference to a new session, call the standard `HttpServletRequest.getSession()` or `HttpServletRequest.getSession(true)` method.

JSF 5.1.2 Locale

- `public Locale getLocale();`
- `public void setLocale(Locale locale);`

When the `FacesContext` for this request is created, the JSF implementation will call `setLocale()` to record the locale of the current user (either stored in the user's session, if any, or the one returned by calling `getLocale()` on the current servlet request). This `Locale` may be used to support localized decoding and encoding operations. At any time during the request processing lifecycle, `setLocale()` be called to modify the localization behavior for the remainder of the current request.

JSF 5.1.3 Component Tree

- `public Tree getTree();`
- `public void setTree(Tree tree);`

During the *Reconstitute Component Tree* phase of the request processing lifecycle, the JSF implementation will identify the component tree (if any) to be used during the inbound processing phases of the lifecycle, and call `setTree()` to establish it.

The `setTree()` method can also be called by the application (during the *Invoke Application* phase of the request processing lifecycle), to change the component tree that will be rendered (during the *Render Response* phase) to a new one. If the application does not do this, the component tree originally created for this request (as modified by the processing during previous phases of the request processing lifecycle) is the one that will get rendered.

JSF 5.1.4 Application Events Queue

[ED. NOTE -- The mechanism for communicating events to the application will change in a future version of this specification. The following describes the current placeholder implementation.]

- `public Iterator getApplicationEvents();`
- `public int getApplicationEventsCount();`

- `public void addApplicationEvent(FacesEvent event);`

During the inbound phases of the request processing lifecycle (but normally during the *Apply Request Values* phase), user interface components or event handlers can queue events (via a call to the `addApplicationEvent()` method) to be handled by the web application. During the *Invoke Application* phase, the JSF implementation will call `getApplicationEvents()` and dispatch each event to the application, via the `ApplicationHandler` instance associated with the `Lifecycle` instance that is processing this request.

JSF.5.1.5 Message Queues

- `public void addMessage(UIComponent component, Message message);`

During the *Apply Request Values*, *Process Validations*, *Update Model Values*, and *Invoke Application* phases of the request processing lifecycle, messages can be queued to either the component tree as a whole (if `component` is `null`), or related to a specific component.

- `public int getMaximumSeverity();`
- `public Iterator getMessages(UIComponent component);`
- `public Iterator getMessages();`

The `getMaximumSeverity()` method returns the highest severity level on any Message that has been queued, regardless of whether or not the message is associated with a specific `UIComponent`. The `getMessages(UIComponent)` methods return an `Iterator` over queued Messages, either those associated with the specified `UIComponent`, or those associated with no `UIComponent` if the parameter is `null`. The `getMessages()` method returns an `Iterator` over all queued Messages, whether or not they are associated with a particular `UIComponent`.

For more information about the `Message` class, see Section JSF.5.2 “Message”.

JSF.5.1.6 Lifecycle Management Objects

[ED. NOTE -- The mechanism for registering event listeners for the application will change in a future version of this specification. The following describes the current placeholder implementation.]

- `public ApplicationHandler getApplicationHandler();`
- `public ViewHandler getViewHandler();`

Return the application handler and view handler instances, respectively, that will be utilized during the *Invoke Application* and *Render Response* phases of the request processing lifecycle, respectively.

JSF.5.1.7 Model Reference Expression Evaluation

- `public Class getModelType(String modelReference);`
- `public Object getModelValue(String modelReference);`
- `public void setModelValue(String modelReference, Object value);`

The expression syntax for model reference expressions corresponds to a variable reference in the expression language defined by JSTL 1.0, but without the “\${” and “}” delimiters.

JSF.5.1.8 FacesEvents Queue

- `public Iterator getFacesEvents();`
- `public void addFacesEvent(FacesEvent event);`

During the inbound phases of the request processing lifecycle up to and including the *Update Model Values* phase, user interface components can queue events (via a call to the `addFacesEvent()` method) to be handled by event listeners registered on the source component, at the end of each request processing phase. The JSF implementation will call `getFacesEvent()` to process the events that have been queued in that phase, as described in Section JSF.2.3 “Common Event Processing”.

JSF.5.1.9 ResponseStream and ResponseWriter

- `public ResponseStream getResponseStream();`
- `public void setResponseStream(ResponseStream responseStream);`
- `public ResponseWriter getResponseWriter();`
- `public void setResponseWriter(ResponseWriter responseWriter);`

JSF supports output that is generated as either a byte stream or a character stream. UIComponents or Renderers that wish to create output in a binary format should call `getResponseStream()` to acquire a stream capable of binary output. Correspondingly, UIComponents or Renderers that wish to create output in a character format should call `getResponseWriter()` to acquire a writer capable of character output.

Due to restrictions of the underlying Servlet APIs, either binary or character output can be utilized for a particular response -- they may not be mixed.

[FIXME - clarify when `setResponseStream`/`setResponseWriter` are called, and by whom]

JSF.5.1.10 Flow Control Methods

- `public void renderResponse();`
- `public void responseComplete();`

Normally, the phases of the request processing lifecycle are executed sequentially, as described in Chapter 2. However, it is possible for components, event listeners, and validators to affect this flow by calling one of these methods.

The `renderResponse()` method signals the JSF implementation that, at the end of the current phase (in other words, after all of the processing normally performed for this phase is completed), control should be transferred immediately to the *Render Response* phase, bypassing any intervening phases that have not yet been performed. For example, an event listener for a tree control that was designed to process UI state changes (such as expanding or contracting a node) on the server would typically call this method to cause the current page to be redisplayed, rather than being processed by the application.

The `responseComplete()` method, on the other hand, signals the JSF implementation that the HTTP response for this request has been completed by some means other than rendering the component tree, and that the request processing lifecycle for this request should be terminated when the current phase is complete. For example, an event listener that decided an HTTP redirect was required would perform the appropriate actions on the response object (i.e. calling `sendRedirect()`) and then call this method.

In some circumstances, it is possible that both `renderResponse()` and `responseComplete()` might have been called for the request. In this case, the JSF implementation must respect the `responseComplete()` call (if it was made) before checking to see if `renderResponse()` was called.

JSF.5.1.11 Access To The Current FacesContext Instance

- `public static FacesContext getCurrentInstance();`
- `protected static void setCurrentInstance(FacesContext context);`

Under most circumstances, JSF components, and application objects that access them, are passed a reference to the `FacesContext` instance for the current request. However, in some cases, no such reference is available. The `getCurrentInstance()` method may be called by any Java class in the current web application to retrieve an instance of the `FacesContext` for this request. The JSF implementation must ensure that this value is set correctly before `FacesContextFactory` returns a `FacesContext` instance, and that the value is maintained in a thread-safe manner.

JSF.5.2 Message

Each message queued within a `FacesContext` is an instance of the `javax.faces.context.Message` abstract class. The following method signatures are supported to retrieve the properties of the completed message:

- `public String getDetail();`
- `public int getSeverity();`
- `public String getSummary();`

The message properties are defined as follows:

- *detail* - Localized detail text for this `Message` (if any). This will generally be additional text that can help the user understand the context of the problem being reported by this `Message`, and offer suggestions for correcting it.
- *severity* - An integer value defining how serious the problem being reported by this `Message` instance should be considered. Four standard severity values (`SEVERITY_INFO`, `SEVERITY_WARN`, `SEVERITY_ERROR`, and `SEVERITY_FATAL`) are defined as symbolic constants in the `Message` class.
- *summary* - Localized summary text for this `Message`. This is normally a relatively short message that concisely describes the nature of the problem being reported by this `Message`.

JSF.5.3 MessageImpl

`MessageImpl` is a concrete implementation of `Message` that can serve as a convenient base class for messages provided by JSF implementations, component libraries, or by applications. It offers the following additional signatures above those defined by `Message`.

- `public MessageImpl();`
- `public MessageImpl(int severity, String summary, String detail);`

These constructors support the creation of uninitialized and initialized `Message` instances, respectively.

- `public void setDetail(String detail);`
- `public void setSeverity(int severity);`
- `public void setSummary(String summary);`

These property setters support modification of the properties of the `MessageImpl` instance.

JSF.5.4 ResponseStream

`ResponseStream` is an abstract class representing a binary output stream for the current response. **FIXME** - It has exactly the same method signatures as the `java.io.OutputStream` class.

JSF.5.5 ResponseWriter

`ResponseWriter` is an abstract class representing a character output stream for the current response. It supports both low-level and high level APIs for writing character based information.

- `public void close() throws IOException;`
- `public void flush() throws IOException;`
- `public void write(char c[]) throws IOException;`
- `public void write(char c[], int off, int len) throws IOException;`
- `public void write(int c) throws IOException;`
- `public void write(String s) throws IOException;`
- `public void write(String s, int off, int len) throws IOException;`

The `ResponseWriter` class extends `java.io.Writer`, and therefore inherits these method signatures for low-level output. The `close()` method flushes the underlying output writer, and causes any further attempts to output characters to throw an `IOException`. The `flush()` method flushes any buffered information to the underlying output writer, and commits the response. The `write()` methods write raw characters directly to the output writer.

[**FIXME** - The existence of the following methods, here rather than on a separate wrapper class, needs to be reviewed. It causes complexity in the concrete implementations `JspResponseWriter` and `ServletResponseWriter`.]

- `public void startDocument() throws IOException;`
- `public void endDocument() throws IOException;`

Write appropriate characters at the beginning (`startDocument()`) or end (`endDocument()`) of the current response.

- `public void startElement(String name) throws IOException;`

Write the beginning of a markup element (the “<” character followed by the element name), which causes the `ResponseWriter` implementation to note internally that the element is open. This can be followed by zero or more calls to `writeAttribute()` or `writeURIAttribute()` to append an attribute name and

value to the currently open element. The element will be closed (i.e. the trailing ">" added) on any subsequent call to `startElement()`, `writeComment()`, `writeText()`, or `endDocument()`.

- `public void endElement(String name) throws IOException;`

Write a closing for the specified element, closing any currently opened element first if necessary.

- `public void writeComment(Object comment) throws IOException`

Write a comment string wrapped in appropriate comment delimiters, after converting the comment object to a String first. Any currently opened element is closed first.

- `public void writeAttribute(String name, Object value) throws IOException;`
- `public void writeURIAttribute(String name, Object value) throws IOException;`

These methods add an attribute name/value pair to an element that was opened with a previous call to `startElement()`, throwing an exception if there is no currently open element. The `writeAttribute()` method causes character encoding to be performed in the same manner as that performed by the `writeText()` methods. The `writeURIAttribute()` method assumes that the attribute value is a URI, and performs URI encoding (such as "%" encoding for HTML).

- `public void writeText(Object text) throws IOException;`
- `public void writeText(char text) throws IOException;`
- `public void writeText(char text[]) throws IOException;`
- `public void writeText(char text[], int off, int len) throws IOException;`

Write text (converting from Object to String first, if necessary), performing appropriate character encoding. Any currently open element created by a call to `startElement()` is closed first.

JSF.5.6 MessageResources

The abstract class `javax.faces.context.MessageResources` represents a mechanism by which localized messages associated with a unique (per `MessageResources` instance) message identifier. JSF implementations make multiple `MessageResources` instances available via a `MessageResourcesFactory` (see the next section for details).

- `public Message getMessage(FacesContext context, String messageId);`

```
■ public Message getMessage(FacesContext context, String
messageId, Object params[]);
```

Return a localized¹ Message instance for the specified message identifier, optionally modified by substitution parameters² in the second method signature. If the specified message identifier is not recognized by this MessageResources instance, these methods return null instead of a Message instance.

- public Message getMessage(FacesContext context, String
messageId, Object param0);
- public Message getMessage(FacesContext context, String
messageId, Object param0, Object param1);
- public Message getMessage(FacesContext context, String
messageId, Object param0, Object param1, Object param2);
- public Message getMessage(FacesContext context, String
messageId, Object param0, Object param1, Object param2,
Object param3);

Convenience for creating messages based on one, two, three, or four substitution parameters.

JSF.5.7 MessageResourcesFactory

A single instance of javax.faces.context.MessageResourcesFactory must be made available to each JSF-based web application running in a servlet container. The factory instance can be acquired by JSF implementations, or by application code, by executing:

```
MessageResourcesFactory factory = (MessageResourcesFactory)
    FactoryFinder.getFactory
    (FactoryFinder.MESSAGE_RESOURCES_FACTORY);
```

The MessageResourcesFactory implementation class supports the following methods:

- public MessageResources getMessageResources(String
messageResourcesId);

This method creates (if necessary) and returns a MessageResources instance. All requests for the same message resources identifier, from within the same web application, will return the same MessageResources instance, which must be programmed in a thread-safe manner.

Every JSF implementation must provide two MessageResources instances, associated with the message resources identifiers named by the following String constants:

1. Localization is performed relative to the locale property of the specified FacesContext.
2. Typical MessageResources implementations support message template strings that can be passed to instances of java.text.MessageFormat, with placeholders like “{0}” for inserting substitution parameters.

- **MessageResourcesFactory.FACES_API_MESSAGES** - Identifier for a MessageResources instance defining the message identifiers used in javax.faces.* classes defined in this specification (such as the predefined Validator implementations).
- **MessageResourcesFactory.FACES_IMPL_MESSAGES** - Identifier for a MessageResources instance defining the message identifiers used in the classes comprising a JSF implementation.

Additional MessageResources instances can be registered at any time (see below).

- `public Iterator getMessageResourcesIds();`

Return an Iterator over the message resource identifiers for all MessageResources implementations available via this MessageResourcesFactory. This Iterator must include the standard message resources identifiers described earlier in this section.

- `public void addMessageResources(String messageResourcesId, MessageResources messageResources);`

Register an additional MessageResources instance under the specified message resources identifier. This method may be called at any time by JSF implementations, JSF-based applications, and third party component libraries utilized by an application. Once registered, the MessageResources instance is available for the remainder of the lifetime of this web application.

JSF 5.8 FacesContextFactory

A single instance of javax.faces.context.FacesContextFactory must be made available to each JSF-based web application running in a servlet container. This class is primarily of use by JSF implementors-- applications will not generally call it directly. The factory instance can be acquired, by JSF implementations or by application code, by executing:

```
FacesContextFactory factory = (FacesContextFactory)
    FactoryFinder.getFactory
        (FactoryFinder.FACES_CONTEXT_FACTORY);
```

The FacesContextFactory implementation class provides the following method signature to create (or recycle from a pool) a FacesContext instance:

- `public FacesContext getFacesContext(ServletContext context, ServletRequest request, ServletResponse response, Lifecycle lifecycle);`

Create (if necessary) and return a FacesContext instance that has been configured based on the specified parameters.

JSF.5.9 Tree

As mentioned in Section JSF.3.1.4 “Component Tree Manipulation”, JavaServer Faces supports the ability to combine multiple `UIComponent` instances into a tree structure, with a single component as the root node. Beyond this, JSF also supports the ability to create component trees dynamically (using the `TreeFactory` API described in the following section), with preconfigured components and other attributes. These trees are represented by instances of the abstract class `javax.faces.tree.Tree` with the following method signatures:

- `public String getRenderKitId();`
- `public void setRenderKitId(String renderKitId);`

The `RenderKit` instance associated with a `Tree` is used as a factory for `Renderer` instances for components that are configured with a non-null `rendererType` property for delegated implementation of decode and encode operations. Component trees will often be constructed with a default `RenderKit` provided by the JSF implementation. In addition, JSF-based applications can choose the `RenderKit` that will be used to create a particular response, so that the same components can be rendered in different ways for different clients.

- `public UIComponent getRoot();`

This method returns the root node of the component tree associated with this `Tree` instance.

- `public String getTreeId();`

This method returns the *tree identifier* of the component tree associated with this `Tree` instance. This identifier is utilized in two different phases of the request processing lifecycle:

- During the *Reconstitute Component Tree* phase, an appropriate tree identifier is extracted from the incoming request as part of the state information being created from the incoming request data. This tree will later be updated by subsequent phases of request processing.
- During the *Invoke Application Handler* phase, JSF-based applications may create a tree that corresponds to the output page to be created by this response¹. For more information, see Section JSF.2.4.2 “Create And Configure A New Component Tree”.

Tree identifiers must be composed of characters that are legal in URLs, optionally including slash (‘/’) characters.

1. Unless the application does this, the response will be based on the same tree as the one manipulated by previous request processing lifecycle phases.

JSF.5.10 TreeFactory

[FIXME - The functionality described below is not required if we are dispensing with the idea of external metadata for a component tree. The remaining need is a way to portably instantiate Tree instances with a given tree identifier and RenderKit.]

A single instance of javax.faces.tree.TreeFactory must be made available to each JSF-based web application running in a servlet container. The factory instance can be acquired, by JSF implementations or by application code, by executing:

```
TreeFactory factory = (TreeFactory)
    FactoryFinder.getFactory(FactoryFinder.TREE_FACTORY);
```

The TreeFactory implementation class provides the following method:

- `public Tree getTree(FacesContext context, String treeId)
throws FacesException;`

This method instantiates a Tree instance, preconfigured with a default renderKitId provided by the JSF implementation.

JSF.6

Lifecycle Management

In Chapter 2, the required functionality of each phase of the request processing lifecycle was described. This chapter describes the standard APIs used by JSF implementations to manage and execute the lifecycle. Each of these classes and interfaces is part of the `javax.faces.lifecycle` package.

Page authors, component writers, and application developers, in general, will not need to be aware of the lifecycle management APIs -- they are primarily of interest to tool providers and JSF implementors.

JSF.6.1 Lifecycle

Upon receipt of each JSF-destined request to this web application, the JSF implementation must acquire a `Lifecycle` instance with which to manage the request processing lifecycle, as described in Section JSF.6.5 “LifecycleFactory”. The `Lifecycle` instance acts as a state machine, and invokes appropriate processing logic to implement the required functionality for each phase of the request processing lifecycle, as described in Section JSF.2.2 “Standard Request Processing Lifecycle Phases”.

- `public void execute(FacesContext context) throws FacesException;`
- `public int executePhase(FacesContext context, Phase phase) throws FacesException;`

[ED. NOTE -- The `executePhase()` method will likely be removed in a future version of this specification.]

The JSF implementation must call the `execute()` method to perform the entire request processing lifecycle, once it has acquired an appropriate `FacesContext` for this request as described in Section JSF.2.4.1 “Acquire Faces Object References”. This method must execute all phases of the request processing lifecycle, according to the requirements of Section JSF.2.2 “Standard Request Processing Lifecycle Phases”.

- `public ApplicationHandler getApplicationHandler();`
- `public void setApplicationHandler(ApplicationHandler handler);`

[ED. NOTE -- The mechanisms for passing events to application components will be revised in a future version of this specification.]

In order to process application events during the *Invoke Application* phase of the request processing lifecycle, the application must register an instance of a class that implements the `ApplicationHandler` interface, as described in Section JSF.6.3 “`ApplicationHandler`”.

Unless overridden (by a call to `setApplicationHandler()` before the first request is processed), the JSF implementation must provide a default `ApplicationHandler` implementation that throws an `IllegalStateException` indicating that no application handler has been configured.

- `public ViewHandler getViewHandler();`
- `public void setViewHandler(ViewHandler handler);`

A `ViewHandler` (see Section JSF.6.4 “`ViewHandler`”) is a pluggable mechanism for performing the required processing to transmit the component tree to the `ServletResponse` that is associated with the `FacesContext` for a request, during the *Render Response* phase of the request processing lifecycle.

Unless overridden (by a call to `setViewHandler()` before the first request is processed), the JSF implementation must provide a default `ViewHandler` implementation that converts the `treeId` property of the component tree into a context-relative resource path, and performs a `RequestDispatcher.forward()` to the corresponding resource.

JSF.6.2 Phase

[ED. NOTE -- This class is likely to be removed from a future version of this specification, because there is no need for a JSF-based application to utilize it.]

The standard processing performed by each phase of the request processing lifecycle, as described in Chapter 2, must be implemented in a subclass of `Phase` that is provided by the JSF implementation.

- `public int execute(FacesContext context) throws FacesException;`

Perform the functionality required by the specification of the current phase of the request processing lifecycle, and return a *state change indicator* that indicates whether and where lifecycle processing should continue. The following symbolic constants define the valid return values, and their corresponding meanings:

- `Phase.GOTO_EXIT` - Indicates that the entire request processing lifecycle has been completed, and no further `Phase` instances should be invoked. This value is normally returned only from the `Phase` instance that implements the *Render Response* phase; however, it must be returned from **any** `Phase` instance that completes the creation of the servlet response for the current request (for example, by performing an HTTP redirect).
- `Phase.GOTO_NEXT` - Indicates that processing should proceed with the next registered `Phase` instance in the normal sequence. This is the typical value, and should be returned in all cases other than when returning `GOTO_EXIT` or `GOTO_RENDER` is appropriate.
- `Phase.GOTO_RENDER` - Indicates that processing should immediately proceed to the *Render Response* phase, skipping any intervening `Phase` instances. For example, this return value will be used when the *Process Validations* phase has detected one or more validation errors, and wishes to bypass the *Update Model Values* and *Invoke Application* phases in order to redisplay the current page.

JSF.6.3 ApplicationHandler

[ED. NOTE -- The behavior of this API will be revised in a future version of this specification. The current mechanism is a placeholder until a more sophisticated interface design is completed.]

A JSF-based application must register a class that implements the `ApplicationHandler` interface in order to process application events during the *Invoke Application* phase of the request processing lifecycle. This is typically done at application startup time, by utilizing the `LifecycleFactory` described in Section FIXME to create the `Lifecycle` instance that will be utilized, and then calling its `setApplicationHandler()` method.

- `public boolean processEvent(FacesContext context, FacesEvent event);`

During the *Invoke Application* phase of the request processing lifecycle, the JSF implementation will call the `processEvent()` method on the registered `ApplicationHandler` instance for each `FacesEvent` that has been queued in the `FacesContext`, until either an event handler returns `true` (indicating a desire to proceed immediately to the *Render Response* phase of the request processing lifecycle), or until all queued application events have been processed.

During application event processing, the application may perform functional actions required by the application's logic. In particular, it may decide that it wishes to display a response page other than the one that corresponds to the component tree, and perform the following steps described in Section JSF.2.4.2 "Create And Configure A New Component Tree".

JSF.6.4 ViewHandler

A JSF implementation must register an implementation of the `ViewHandler` interface for use during the *Render Response* phase of the request processing lifecycle. Prior to the first request being processed by a `Lifecycle` instance, an alternative `ViewHandler` implementation may be registered by a call to the `setViewHandler()` method.

- `public void renderView(FacesContext context) throws IOException, ServletException;`

Perform whatever actions are required to render the component tree to the `ServletResponse` associated with the specified `FacesContext`.

JSF.6.5 LifecycleFactory

A single instance of `javax.faces.lifecycle.LifecycleFactory` must be made available to each JSF-based web application running in a servlet container. This class is primarily of use to tools providers -- applications will not generally call it directly. The factory instance can be acquired by JSF implementations or by application code, by executing:

```
LifecycleFactory factory = (LifecycleFactory)
    FactoryFinder.getFactory(FactoryFinder.LIFECYCLE_FACTORY);
```

The `LifecycleFactory` implementation class supports the following methods:

- `public void addLifecycle(String lifecycleId, Lifecycle lifecycle);`

Register a new `Lifecycle` instance under the specified lifecycle identifier, and make it available via calls to `getLifecycle()` for the remainder of the current web application's lifetime.

```
■ public Lifecycle getLifecycle(String lifecycleId);
```

The `LifecycleFactory` implementation class provides this method to create (if necessary) and return a `Lifecycle` instance. All requests for the same lifecycle identifier from within the same web application will return the same `Lifecycle` instance, which must be programmed in a thread-safe manner.

Every JSF implementation must provide a `Lifecycle` instance for a default lifecycle identifier that is designated by the String constant `LifecycleFactory.DEFAULT_LIFECYCLE`. For advanced uses, a JSF implementation may support additional lifecycle instances, named with unique lifecycle identifiers.

- `public Iterator<String> getLifecycleIds();`

This method returns an iterator over the set of lifecycle identifiers supported by this factory. This set must include the value specified by `LifecycleFactory.DEFAULT_LIFECYCLE`.

JSF.7

Rendering Model

JavaServer Faces supports two programming models for decoding component values from incoming requests, and encoding component values into outgoing responses - the *direct implementation* and *delegated implementation* models. When the *direct implementation* model is utilized, components must decode and encode themselves. When the *delegated implementation* programming model is utilized, these operations are delegated to a Renderer instance associated (via the `rendererType` property) with the component. This allows applications to deal with components in a manner that is predominantly independent of how the component will appear to the user, while allowing a simple operation (selection of a particular RenderKit) to customize the decoding and encoding for a particular client device or localized application user.

Component writers, application developers, tool providers, and JSF implementations will often provide one or more RenderKit implementations (along with a corresponding library of Renderer instances). In many cases, these classes will be provided along with the `UIComponent` classes for the components supported by the RenderKit. Page authors will generally deal with RenderKits indirectly, because they are only responsible for selecting a render kit identifier to be associated with a particular page, and a `rendererType` property for each `UIComponent` that is used to select the corresponding Renderer.

JSF.7.1 RenderKit

A RenderKit instance is optionally associated with a component tree, and supports the *delegated implementation* programming model for the decoding and encoding of component values. Each JSF implementation must provide a default RenderKit instance (named by the render kit identifier associated with the String constant `RenderKitFactory.DEFAULT_RENDER_KIT` as described below) that is utilized if no other RenderKit is selected.

```
■ public Iterator getComponentClasses();
```

Return an `Iterator` over the set of `UIComponent` subclasses known to be supported by the `Renderers` registered with this `RenderKit`. The set of classes returned by this method are not guaranteed to be the complete set of component classes supported (i.e. component classes for which one of the included `Renderers` would return `true` from a call to a `supportsComponentType()` method). However, `RenderKit` instances are encouraged to explicitly enumerate the component classes that are associated with its `Renderers`, to assist development tools in providing more information in their user interfaces.

```
■ public Renderer getRenderer(String rendererType);
```

Create (if necessary) and return a `Renderer` instance corresponding to the specified `rendererType`, which will typically be the value of the `rendererType` property of a `UIComponent` about to be decoded or encoded.

```
■ public Iterator getRendererTypes(UIComponent component);  
■ public Iterator getRendererTypes(String componentType);
```

Return an `Iterator` over the set of renderer types for registered `Renderer` instances (if any) that are managed by this `RenderKit` instance and know how to support components of the specified component class or type.

```
■ public Iterator getRendererTypes();
```

Return an `Iterator` over all of the renderer types for the `Renderers` registered with this `RenderKit` instance.

```
■ public void addComponentClass(Class componentClass);  
■ public void addRenderer(String rendererType, Renderer renderer);
```

Applications that wish to go beyond the capabilities of the standard `RenderKit` that is provided by every JSF implementation may either choose to create their own `RenderKit` instances and register them with the `RenderKitFactory` instance (see Section JSF 7.3 “`RenderKitFactory`”), or integrate additional supported component classes and/or `Renderer` instances into an existing `RenderKit` instance. For example, it will be common to for an application that requires custom component classes and `Renderers` to register them with the standard `RenderKit` provided by the JSF implementation, at application startup time.

JSF 7.2 Renderer

A `Renderer` instance implements the decoding and encoding functionality of components, during the *Apply Request Values* and *Render Response* phases of the request processing lifecycle, when the component has a non-null value for the `rendererType` property.

```
■ public void decode(FacesContext context, UIComponent component) throws IOException;
```

For components utilizing the *delegated implementation* programming model, this method will be called during the *Apply Request Values* phase of the request processing lifecycle, for the purpose of converting the incoming request information for this component back into a new local value. See the Javadocs for the `Renderer.decode()` method for details on its responsibilities.

- `public void encodeBegin(FacesContext context, UIComponent component) throws IOException;`
- `public void encodeChildren(FacesContext context, UIComponent component) throws IOException;`
- `public void encodeEnd(FacesContext context, UIComponent component) throws IOException;`

For components utilizing the *delegated implementation* programming model, these methods will be called during the *Render Response* phase of the request processing lifecycle. These methods have the same responsibilities as the corresponding `encodeBegin()`, `encodeChildren()`, and `encodeEnd()` methods of `UIComponent` (described in Section JSF.3.1.9 “Component Specialization Methods” and the corresponding Javadocs) when the component implements the *direct implementation* programming model.

- `public boolean supportsComponentType(UIComponent component);`
- `public boolean supportsComponentType(String componentType);`

These methods return `true` if the specified component class or type is supported by this `Renderer` instance. For supported components, the `getAttributeNames()` and `getAttributeDescriptor()` methods can be utilized to acquire metadata information useful in configuring the attributes of these components, when it is known that this particular `Renderer` instance will be utilized.

- `public Iterator getAttributeNames(UIComponent component);`
- `public Iterator getAttributeNames(String componentType);`

Return an `Iterator` over the attribute names supported by this `Renderer` for the specified component class or type. These methods are useful to tool providers in building user interfaces to configure the properties and attributes of a particular component, when it is known that this particular `Renderer` will be utilized.

- `public AttributeDescriptor getAttributeDescriptor(UIComponent component, String name);`
- `public AttributeDescriptor getAttributeDescriptor(String componentType, String name);`

Return an `AttributeDescriptor` for the specified attribute name, as supported for the specified component class or type. These methods are useful to tool providers in building user interfaces to configure the properties and attributes of a particular component, when it is known that this particular `Renderer` will be utilized.

JSF.7.3 RenderKitFactory

A single instance of `javax.faces.render.RenderKitFactory` must be made available to each JSF-based web application running in a servlet container. The factory instance can be acquired by JSF implementations, or by application code, by executing

```
RenderKitFactory factory = (RenderKitFactory)
    FactoryFinder.getFactory(FactoryFinder.RENDER_KIT_FACTORY);
```

The `RenderKitFactory` implementation class supports the following methods:

- `public RenderKit getRenderKit(String renderKitId);`
- `public RenderKit getRenderKit(String renderKitId, FacesContext context);`

This method creates (if necessary) and returns a `RenderKit` instance. All requests for the same render kit identifier will return the same `RenderKit` instance, which must be programmed in a thread-safe manner.

Every JSF implementation must provide a `RenderKit` instance for a default render kit identifier that is designated by the String constant

`RenderKitFactory.DEFAULT_RENDER_KIT`. Additional render kit identifiers, and corresponding instances, can also be made available.

- `public Iterator getRenderKitIds();`

This method returns an `Iterator` over the set of render kit identifiers supported by this factory. This set must include the value specified by `RenderKitFactory.DEFAULT_RENDER_KIT`.

- `public void addRenderKit(String renderKitId, RenderKit renderKit);`

At any time, additional `RenderKit` instances, and their corresponding identifiers, can be registered by the JSF implementation, by included component libraries, or by applications themselves. Once a `RenderKit` instance has been registered, it may be associated with a component tree by calling the `setRenderKit()` method of the corresponding `Tree` instance.

JSF.7.4 Standard HTML RenderKit Implementation

To ensure application portability, all JSF implementations are required to include support for a `RenderKit`, and the associated `Renderers`, that meet the requirements defined in this section, to generate textual markup that is compatible with HTML/4.01. JSF implementors, and other parties, may also provide additional `RenderKit` libraries, or additional `Renderers` that are added to the standard `RenderKit` at application startup time, but applications must ensure that the standard `Renderers` are made available for the web application to utilize them.

The JavaBean properties of standard component classes can be considered to be *render-independent* attributes of the components -- indeed, their values are available through the standard `getAttribute()` method or via property getter method calls. These values, as the term implies, are independent of the rendering technology that will be used to create the user interface. For example, a `UICommand` component has a `commandName` render-independent attribute that identifies the application command that should be performed when this command is selected. A `UICommand` can be displayed in many different ways, using many different rendering technologies -- but the *meaning* of a command name does not change.

Page authors can configure the precise details of the rendering that should be performed for JSF components by setting *render-dependent* attributes on those components, and by selecting a specific Renderer (from the RenderKit associated with this response) to be utilized by specifying the `rendererType` attribute¹. As the term implies, these attributes are meaningful only to the Renderer actually selected (via a choice of RenderKit, and configuration of a `rendererType` attribute on each component) to perform the encoding and decoding processing for a particular interaction. Attributes that are not recognized by a particular Renderer are simply ignored.

The set of render-dependent attribute names and values will typically have a close correspondence to concepts in the markup language or technology used to actually create the response content, and to read the subsequent request. Indeed, this is the case for the Standard HTML RenderKit implementation -- to maximize familiarity for users already familiar with HTML, most of the render-dependent attribute names recognized by the library have a one-to-one correspondence with attribute names on the corresponding HTML elements that will be created.

The required Renderers are organized into subsections based on the standard `UIComponent` class supported by those Renderers. Many of these Renderers share common attributes (or sets of attributes) -- to minimize the repetitiveness of the Renderer descriptions, the common rules are documented here.

TABLE 7-1 Commonly Used Render-Dependent Attributes

Attribute Name	Description
<code>columnClasses</code>	A comma-delimited list of CSS style classes that will be used in sequence for the column elements of a table-oriented layout (using the HTML <code>class</code> attribute). If the rendered table has more columns than the number of class names in this list, the list will be restarted from the beginning as many times as necessary; however, the first column of a new row will always be assigned the first style class in this list.
<code>converter</code>	Either a String that contains the <i>converter id</i> of a Converter instance to be looked up in theConverterFactory for this web application, or a Converter instance to be used directly, as described in Section JSF.3.3 “Conversion Model”.

1. For JSP page authors that use the Standard HTML RenderKit Tag Library described in Section FIXME/8.6, the `rendererType` setting is made implicitly by virtue of the action name selected for each component.

TABLE 7-1 Commonly Used Render-Dependent Attributes

Attribute Name	Description
dateStyle	Date style pattern name as defined by the <code>java.text.DateFormat</code> class. Valid values: short, medium, long, or full.
formatPattern	String version of a formatting pattern suitable for use with a <code>java.text.DateFormat</code> or <code>java.text.NumberFormat</code> instance.
numberStyle	Number style supported by the <code>java.text.NumberFormat</code> class. Valid values are: currency, integer, number, or percent.
rowClasses	A comma-delimited list of CSS style classes that will be used in sequence for the row elements of a table-oriented layout (using the HTML <code>class</code> attribute). If the rendered table has more columns than the number of class names in this list, the list will be restarted from the beginning as many times as necessary.
timeStyle	Time style pattern name as defined by the <code>java.text.DateFormat</code> class. Valid values) short, medium, long, or full.
timezone	A time zone identifier as defined by the <code>java.util.TimeZone</code> class.
xxxClass	A CSS style class to be used when rendering this component, where “xxx” corresponds to the component class name without the “UI” prefix, and converted to lower case ¹ .

1. In general, component attributes passed directly to the HTML output will have the same name as the corresponding HTML attribute, so one would expect an attribute named “class” to be used for this purpose. However, due to technical restrictions in JSP pages, this attribute name is difficult to use (requires a BeanInfo class). Therefore, an easy-to-remember convention for the CSS style class name has been substituted.

FIXME -- figure out a way to consisely document the attributes that correspond 1:1 with HTML/4.01 attributes and are simply passed through.

JSF 7.4.1 Renderer Types for UICommand Components

TABLE 7-1 Renderer Types for UICommand Components

Renderer Type	Data Type	Render Dependent Attributes	Decode Behavior	Encode Behavior
Button	String	label,type	Submit FormEvent to application as per default UICommand behavior.	Render an HTML <input> element with the specified type (submit, reset). The button text is specified literally by the label attribute.
Button	String	key, bundle, type	Submit FormEvent to application as per default UICommand behavior.	Render an HTML <input> element with the specified type (submit, reset). The button text is localized based on the key and bundle attributes.
Hyperlink	String	(none)	Submit CommandEvent to application.	Render an HTML <a> element that generates a hyperlink back to the tree identifier of the tree, so that a command event can be enqueued ¹ .
Hyperlink	String	href	(none)	Render an HTML <a> element that links to the specified URL. Nested UIParameter components (with the name attribute defined) can be used to configure request parameters that will be added to the specified URL. URL rewriting will be performed to maintain session state in the absence of cookies.

1. The HTML rendered by this Renderer may require the use of client-side JavaScript if the JSF implementation is maintaining component tree state information as part of the page, instead of keeping it on the server side.

A UICommand component (see Section JSF 4.1 “UICommand”) is intended to represent a user interface component that a user can “activate” in some manner, which will trigger some application defined action. For HTML, two common methods of rendering a UICommand are supported:

- As an HTML <input> element, used for submit and reset buttons nested inside a <form> element. Triggering such a button will cause a FormEvent to be submitted to the application, identifying which form and which submit button were submitted.

- As an HTML <a> (hyperlink) element, used to forward control to either submit a CommandEvent to the application (if no href attribute is specified), or forward control to an arbitrary URL (if href is specified).

The following requirements apply to the “Hyperlink” Renderer described in this section:

- Nested `UIGraphic` and `UIOutput` components can be used to provide the image background and/or text of the rendered button or hyperlink.

JSF 7.4.2 Renderer Types for UIForm Components

TABLE 7-1 Renderer Types for UIForm Components

Renderer Type	Data Type	Render Dependent Attributes	Decode Behavior	Encode Behavior
Form	(none)		Per specified default behavior for the UIForm decode() method.	Per the specified default behavior for the UIForm encodeXxx() methods.

The `UIForm` component (see Section JSF 4.2 “UIForm”) contains a set of `UIInput` components representing request parameters that will be included when the form is submitted, as well as `UICommand` components that represent submit and reset buttons. It will be rendered as an HTML <form> element, and all nested child components will be themselves rendered in between the beginning and ending elements of the form.

It will be quite common to manage the generated layout of a form by nesting a `UIPanel` component inside to generate the required layout information. For example, consider a component tree as follows, with increasing levels of indentation representing child components of the immediately preceding component:

`UIForm` component with `formName="logonForm"`

- `UIPanel` component with `rendererType="Grid"` and `columns="2"`.
 - `UIOutput` component with `rendererType="Label"` for the username prompt.
 - `UIInput` component with `rendererType="Text"` for the username field
 - `UIOutput` component with `rendererType="Label"` for the password prompt.
 - `UIInput` component with `rendererType="Secret"` for the password field.

The “Grid” renderer will generate an HTML <table> element with two columns per row, and lay its nested components out inside <td> elements automatically, without the page author needing to be aware of how this is done. In addition, the

appearance of the generated table can be managed using CSS stylesheets, by using the appropriate render-dependent attributes defined by the “Grid” renderer (see Section JSF.7.4.6 “Renderer Types for UIPanel Components”).

JSF.7.4.3 Renderer Types for UIGraphic Components

TABLE 7-1 Renderer Types for UIGraphic Components

Renderer Type	Data Type	Render Dependent Attributes	Decode Behavior	Encode Behavior
Image	String			Render an HTML element with a <code>src</code> attribute set from the current value of the underlying component. URL rewriting is performed to maintain session state in the absence of cookies.
Image		key, bundle		Render an HTML element with a <code>src</code> attribute localized from the resource bundle. URL rewriting will be performed to maintain session state in the absence of cookies.

The `UIGraphic` component (see Section JSF.4.3 “UIGraphic”) represents an output-only display of a graphical image. For HTML, the “Image” Renderer creates an `` element whose `src` attribute is either the current value of this component (no `key` attribute present), or is a localized value looked up in an appropriate resource bundle (`key` attribute present).

JSF 7.4.4 Renderer Types for UIInput Components

TABLE 7-1 Renderer Types for UIInput Components

Renderer Type	Data Type	Render Dependent Attributes	Decode Behavior	Encode Behavior
Date	Date, long	dateStyle, timezone	Get DateFormat instance as for encoding, parse text input, convert as necessary and store.	Get current value, convert to Date, create localized DateFormat date instance for the specified dateStyle, set time zone (if specified), render formatted result as an HTML <input> element of type "text".
DateTime	Date, long	formatPattern, timezone	Get DateFormat instance as for encoding, parse text input, convert as necessary and store.	Get current value, convert to Date, create localized SimpleDateFormat based on formatPattern, set time zone (if specified), render formatted result as an HTML <input> element of type "text".
DateTime	Date, long	dateStyle, timeStyle, timezone	Get DateFormat instance as for encoding, parse text input, convert as necessary and store.	Get current value, convert to Date, create localized DateFormat date/time instance set time zone (if specified), render formatted result as an HTML <input> element of type "text".
Hidden	Any	converter	Convert as necessary and store.	Get current value, convert to String, render as an HTML <input> element of type "hidden".
Number	numeric	formatPattern	Get NumberFormat instance as for encoding, parse text input, convert as necessary and store.	Get current value, convert to numeric (as appropriate), create localized NumberFormat instance based on numberStyle, render formatted result as an HTML <input> element of type "text". ¹
Number	numeric	numberStyle	Get NumberFormat instance as for encoding, parse text input, convert as necessary, and store.	Get current value, convert to numeric (as appropriate), create localized NumberFormat instance based on numberStyle, render as an HTML <input> element of type "text".

TABLE 7-1 Renderer Types for UIInput Components

Renderer Type	Data Type	Render Dependent Attributes	Decode Behavior	Encode Behavior
Secret	Any	converter, redisplay	Convert as necessary and store.	Get current value, convert to String, render as an HTML <input> element of type “password”. The <code>redisplay</code> attribute is a boolean flag (default=false) indicating whether or not the current value should be displayed ² .
Text	Any	converter	Convert as necessary and store.	Get current value, convert to String, render as an HTML <input> element of type “text”.
Textarea	Any	converter	Convert as necessary and store.	Get current value, convert to String, render as an HTML <textarea> element.
Time	Date, long	timeStyle, timezone	Get <code>DateFormat</code> instance as for encoding, parse text input, convert as necessary and store.	Get current value, convert to Date, create localized <code>DateFormat</code> time instance for the specified <code>timeStyle</code> , set time zone (if specified), render formatted result as an HTML <input> element of type “text”.

1. **FIXME** - When `numberStyle` is set to `CURRENCY`, we need to specify `currencyCode` and `currencySymbol` like the JSTL `<fmt:formatNumber>` tag does. It may be worth having a separate Renderer Type for this use case.

2. Although such a field displays asterisks on most browsers, the plaintext value is still easily visible when a user uses the “View Source” menu option, and this can be a security risk in some environments.

The `UIInput` component (see Section JSF.4.4 “`UIInput`”) represents a user interface component that both displays information (when a response is being rendered) and updates its current value on a subsequent request (typically a form submit). A variety of `Renderers` are available to deal with common cases involving the display of Strings, numbers, and date/time values in a variety of localized or custom-pattern scenarios.

In HTML, the most common mechanism for rendering `UIInput` components is an `<input>` element of type “text”. For maximum flexibility, the “Text” renderer will accept current values of any type, and perform an appropriate conversion to String (when rendering the response) or from String (when processing the input). The details of conversion can be managed by selecting one of the following approaches:

- If no `converter` attribute is specified, any non-String value will be converted to String by calling its `toString()` method during rendering. On subsequent request processing, no conversion is performed -- the local value will be stored as a String.

- If a converter attribute is specified, it must be either a String (used to look up a Converter instance in theConverterFactory for our web application), or a Converter instance that will be used directly to manage conversions.

[ED. NOTE -- The availability of the standard Converters described in Section JSF 3.3.4 “Standard Converter Implementations” makes the Date, DateTime, Number, and Time renderer types redundant, so they are likely to be removed in a future version of this specification. Feedback is requested on which approach is easier to utilize.]

JSF 7.4.5 Renderer Types for UIOutput Components

TABLE 7-1 Renderer Types for UIOutput Components

Renderer Type	Data Type	Render Dependent Attributes	Decode Behavior	Encode Behavior
Date	Date, long	dateStyle, timezone		Get current value, convert to Date, create localized DateFormat date instance for the specified dateStyle, set time zone (if specified), render formatted result.
DateTime	Date, long	formatPattern, timezone		Get current value, convert to Date, create localized SimpleDateFormat based on formatPattern, set time zone (if specified), render formatted result.
DateTime	Date, long	dateStyle, timeStyle, timezone		Get current value, convert to Date, create localized DateFormat date/time instance set time zone (if specified), render formatted result.
Errors	(none)	clientId		Render the set of error messages (if any) that are associated with the specified client identifier (if clientId is specified), or the global errors not associated with any component (if clientId is a zero-length string), or all errors (if clientId is not specified).
Label	(none)	for		Render the nested components surrounded by an HTML <label> element with the specified for attribute (which must match the rendered id attribute for some input field).
Message	Any	converter		Get current value, convert to String, treat as a MessageFormat pattern (nested UIParameter components may be used to provide values for parameter substitution), render formatted result. ¹

TABLE 7-1 Renderer Types for UIOutput Components

Renderer Type	Data Type	Render Dependent Attributes	Decode Behavior	Encode Behavior
Message	Any	converter, key, bundle		Get localized resource from resource bundle, treat as a MessageFormat pattern (nested UIParameter components may be used to provide values for parameter substitution), render formatted result.
Number	numerical	formatPattern		Get current value, convert to numeric (as appropriate), create localized NumberFormat instance based on numberStyle, render formatted result. ²
Number	numerical	numberStyle		Get current value, convert to numeric (as appropriate), create localized NumberFormat instance based on numberStyle, render formatted result.
Text	Any	converter		Get current value, convert to String, render literally.
Time	Date, long	timeStyle, timezone		Get current value, convert to Date, create localized DateFormat time instance for the specified timeStyle, set time zone (if specified), render formatted result.

1. FIXME - Need to specify an optimization to skip actually processing the MessageFormat when not necessary, such as when there are no {} type parameters in the message text, or when no substitution values are available.

2. FIXME - When numberStyle is set to CURRENCY, we need to specify currencyCode and currencySymbol like the JSTL <fmt:format-Number> tag does. It may be worth having a separate Renderer Type for this use case.

The UIOutput component (see Section JSF.4.5 “UIOutput”) represents a user interface component that only displays information when a response is being rendered. No processing is performed on a subsequent request. A variety of Renderers are available to deal with common cases involving the display of Strings, numbers, and date/time values in a variety of localized or custom-pattern scenarios.

In HTML, it is very common to directly render the current value of the component as a character String. For maximum flexibility, the “Text” renderer will accept current values of any type, and perform an appropriate conversion to String (when rendering the response). The details of conversion can be managed by selecting one of the following approaches:

- If no converter attribute is specified, any non-String value will be converted to String by calling its `toString()` method during rendering.
- If a converter attribute is specified, it must be either a String (used to look up a Converter instance in theConverterFactory for our web application), or a Converter instance that will be used directly to manage conversions.

[ED. NOTE -- The availability of the standard Converters described in Section JSF.3.3.4 “Standard Converter Implementations” makes the Date, DateTime, Number, and Time renderer types redundant, so they are likely to be removed in a future version of this specification. Feedback is requested on which approach is easier to utilize.]

JSF.7.4.6 Renderer Types for UIPanel Components

TABLE 7-1 Renderer Types for UIPanel Components

Renderer Type	Data Type	Render Dependent Attributes	Decode Behavior	Encode Behavior
Data	array, Collection, Map	var		Nested child components represent the template for a row to be generated for each element of a collection represented by the current value of this component. Each element of the collection will be exposed as a request attribute named by var, so that model reference expressions in the template components can access the values for the current element. There should be one child component per column to be created.
Grid	(none)	columnClasses, columns, footerClass, headerClass, rowClasses		Render an HTML <table> element, from our child components, based on the following rules: (a) if headerClass is specified, the first child will be rendered as a header row across all the columns, formatted according to this CSS style class; (b) Each intervening child is rendered in a separate table data element, with a new row started each columns children; (c) if footerClass is specified, the last child will be rendered as a footer row across all the columns, formatting according to this CSS style class.

TABLE 7-1 Renderer Types for UIPanel Components

Renderer Type	Data Type	Render Dependent Attributes	Decode Behavior	Encode Behavior
Group	(none)			Render the child components as specified by their own renderer type settings. This renderer exists to allow the creation of arbitrary groups where a parent component expects to see a single child.
List	(none)	columnClasses, footerClass, headerClass, rowClasses		Render an HTML <table> element, from our child components, based on the following rules: (a) if headerClass is specified, the first child should be a UIPanel with a renderer type of Group, representing the column headers for the table to be rendered; (b) next children should be zero or more UIPanel components with a renderer type of Data that represent the data collection(s) to be iterated over, and whose children represent a template for each row to be rendered; (c) if footerClass is specified, the last child should be a UIPanel with a renderer type of Group, representing the column footers for the table to be rendered.

The UIPanel component (see Section JSF.4.6 “UIPanel”) represents a container for child components whose actual rendering will be managed by the UIPanel component (or its corresponding Renderer) instead of by the components themselves. It is commonly used to compose complex objects out of simple ones. The standard Renderers for HTML manage a variety of complex layout scenarios by embedding appropriate HTML elements (such as <table>, <tr>, and <td>) so that the individual components do not need to worry about these responsibilities.

In many cases (such as the nested components inside a UIPanel using the “Grid” or “List” Renderers, the requirements above require that each cell of the ultimately rendered table be generated by a single child of the outer UIPanel. In many cases, it would be useful to compose the contents of such a cell with an arbitrary set of grandchild components. This can be easily accomplished by using a UIPanel component with a renderer type of “Group” to represent a single component (from the perspective of the parent UIPanel) that has an arbitrarily complex internal structure.

JSF 7.4.7 Renderer Types for UISelectBoolean Components

TABLE 7-1 Renderer Types for UISelectBoolean Components

Renderer Type	Data Type	Render Dependent Attributes	Decode Behavior	Encode Behavior
Checkbox	boolean		Convert as necessary and store.	Get current value, convert to boolean, render as an HTML <input> element of type "checkbox".

The `UISelectBoolean` component (see Section JSF 4.8 “`UISelectBoolean`”) is a specialized subclass of `UIInput` whose value is restricted to being a boolean `true` or `false`. In HTML, such a component is rendered as an HTML `<input>` element of type “checkbox”, and the decode behavior performed on a subsequent form submit will always ensure that the local value of this component reflects the checked or unchecked state of the checkbox in the user interface.

JSF 7.4.8 Renderer Types for UISelectMany Components

TABLE 7-1 Renderer Types for UISelectMany Components

Renderer Type	Data Type	Render Dependent Attributes	Decode Behavior	Encode Behavior
Checkbox	String	layout	Convert as necessary and store.	Get current value, convert to String, get available items from nested <code>UISelectItem</code> and <code>UISelectItems</code> components, render as a series of HTML <code><input></code> elements of type “checkbox”, laid out according to the layout attribute (PAGE_DIRECTION or LINE_DIRECTION).

TABLE 7-1 Renderer Types for UISelectMany Components

Renderer Type	Data Type	Render Dependent Attributes	Decode Behavior	Encode Behavior
Listbox	String[]		Convert as necessary and store.	Get current value(s), convert to String[], get available items from nested UISelectItem and UISelectItems components, render as an HTML <select> element that displays all possible values (so does not need a scrollbar).
Menu	String[]	size	Convert as necessary and store.	Get current value(s), convert to String[], get available items from nested UISelectItem and UISelectItems components, render as an HTML <select> element that displays the number of elements specified by the size attribute (default=1).

The `UISelectMany` component (see Section JSF.4.12 “`UISelectMany`”) represents a user interface component that offers a fixed set of choices to the user, from which zero or more selections can be made. The data value associated with this component should be an array of objects reflecting the currently selected items from the set of available choices. The different Renderers described above offer different choices for how the set of available choices should be displayed.

The set of available choices is constructed by scanning the child components for one or more instances of `UISelectItem` (see Section JSF.4.10 “`UISelectItem`”) or `UISelectItems` (see Section JSF.4.11 “`UISelectItems`”) components. Existence of these child components causes the addition of a single `SelectItem` instance (for a `UISelectItem`) or a set of `SelectItem` instances (for a `UISelectItems`) to the set of available choices. For each available choice, the `itemLabel` property defines the visual appearance of this choice, while the `itemValue` property defines the value that will be added to the array of current selections for this component.

JSF 7.4.9 Renderer Types for UISelectOne Components

TABLE 7-1 Renderer Types for UISelectOne Components

Renderer Type	Data Type	Render Dependent Attributes	Decode Behavior	Encode Behavior
Listbox	String		Convert as necessary and store.	Get current value, convert to String, get available items from nested UISelectItem and UISelectItems components, render as an HTML <select> element that displays all possible values (so does not need a scrollbar).
Menu	String	size	Convert as necessary and store.	Get current value, convert to String, get available items from nested UISelectItem and UISelectItems components, render as an HTML <select> element that displays the number of elements specified by the size attribute (default=1).
Radio	String	layout	Convert as necessary and store.	Get current value, convert to String, get available items from nested UISelectItem and UISelectItems components, render as a series of radio buttons laid out according to the layout attribute (PAGE_DIRECTION, LINE_DIRECTION).

The `UISelectOne` component (see Section JSF 4.13 “`UISelectOne`”) represents a user interface component that offers a fixed set of choices to the user, from which zero or one selections can be made. The data value associated with this component should be an object reflecting the currently selected value from the set of available choices. The different Renderers described above offer different choices for how the set of available choices should be displayed.

The set of available choices is constructed by scanning the child components for one or more instances of `UISelectItem` (see Section JSF 4.10 “`UISelectItem`”) or `UISelectItems` (see Section JSF 4.11 “`UISelectItems`”) components. Existence of these child components causes the addition of a single `SelectItem` instance (for a `UISelectItem`) or a set of `SelectItem` instances (for a `UISelectItems`) to the

set of available choices. For each available choice, the `itemLabel` property defines the visual appearance of this choice, while the `itemValue` property defines the value that will be returned for this component, if the corresponding item is selected.

JSF.8

Integration With JSP

JavaServer Faces implementations must support (although JSF-based applications need not utilize) using JavaServer Pages (JSP) as the page description language for JSF pages. This JSP support is provided by providing custom actions so that a JSF user interface can be easily defined in a JSP page by adding tags corresponding to JSF UI components. A page author should be able to use JSF components in conjunction with the other custom actions (including the JSP Standard Tag Library), as well as standard HTML content and layout embedded in the page.

JSF.8.1 UIComponent Custom Actions

A JSP custom action for a JSF `UIComponent` is constructed by combining properties and attributes of a Java UI component class with the rendering attributes supported by a specific `Renderer` from a concrete `RenderKit`. For example, assume the existence of a concrete `RenderKit`, `HTMLRenderKit`, which supports three `Renderer` types for the `UITextEntry` component:

TABLE 8-1 EXAMPLE RENDERER TYPES

RendererType	Render-Dependent Attributes
“Text”	“columns”
“Secret”	“columns”, “secretChar”
“Multiline”	“columns”, “rows”

The tag library descriptor (TLD) file for the corresponding tag library, then, would define three custom tags -- one per `Renderer`. Below is an example of the tag definition for the “`input_text`” tag¹:

1. This example illustrates a non-normative convention for naming tags based on a combination of the component name and the renderer type. This convention is useful, but not required; custom actions may be given any desired tag name; however the convention is rigorously followed in the Standard HTML RenderKit Tag Library.

```
<tag>
    <name>input_text</name>
    <tagclass>acme.html.tags.InputTag</tagclass>
    <bodycontent>JSP</bodycontent>
    <attribute>
        <name>id</name>
        <required>false</required>
    </attribute>
    <attribute>
        <name>modelReference</name>
        <required>false</required>
    </attribute>
    <attribute>
        <name>columns</name>
        <required>false</required>
    </attribute>
    ...
</tag>
```

Note that the `columns` attribute is derived from the Renderer of type “Text”, while the `id` and `modelReference` attributes are derived from the `UIInput` component class itself. RenderKit implementors will provide a JSP tag library which includes component tags corresponding to each of the component classes (or types) supported by each of the RenderKit’s Renderers. See Section JSF.7.1 “RenderKit” and Section JSF.7.2 “Renderer” for details on the RenderKit and Renderer APIs.

JSF.8.2 Using UIComponent Custom Actions in JSP Pages

The following subsections define how a page author utilizes the custom actions provided by the RenderKit implementor in the JSP pages that create the user interface of a JSF-based web application.

JSF.8.2.1 Declaring the Custom Actions Tag Library

The page author must use the standard JSP `taglib` directive to declare the URI of the tag library to be utilized, as well as the tag prefix used (within this page) to identify tags from this library. For example,

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
```

declares the unique resource identifiers of the tag libraries being used (provided by the supplier of that tag libraries) for the two standard tag libraries that every JSF implementation must provide, as well as the tag prefixes to be used within the current page for referencing actions from these libraries¹.

Tag libraries for UIComponent custom actions can interoperate with additional tag libraries provided by others, including the JSP Standard Tag Library (JSTL), subject to the following restrictions: *FIXME* - document restrictions.

JSF.8.2.2 Defining Components in a Page

A JSF UIComponent custom action can be placed at any desired position in a JSP page that contains the `taglib` directive for the corresponding tag library. For example,

```
<h:input_text id="username" modelReference="logonBean.username" />
```

represents a text entry field, to be displayed with the “Text” renderer type, for the user name field of a logon form component that is nested inside a root component. The `id` attribute specifies the *component id* of a `UIComponent` instance, from within the component tree, to which this tag corresponds. If no `id` is specified, one will be automatically generated by the JSF implementation.

Custom actions that correspond to JSF `UIComponent` instances must subclass either `javax.faces.webapp.FacesTag` (see Section JSF.9.2.4.3 “FacesTag”) or `javax.faces.webapp.FacesBodyTag` (see Section JSF.9.2.4.4 “FacesBodyTag”), depending on whether the tag needs to support `javax.servlet.jsp.tagext.BodyTag` functionality or not.

1. Consistent with the way that namespace prefixes work in XML, the actual prefix used is totally up to the page author, and has no semantic meaning. However, the values shown above are the suggested defaults, are used consistently in tag library examples throughout this specification.

During the *Render Response* phase of the request processing lifecycle, the appropriate encoding methods of a Renderer of the type associated with this tag will be utilized to generate the representation of this component in the response page.

All markup other than UIComponent custom actions is processed by the JSP container, in the usual way. Therefore, you can use such markup to perform layout control, or include non-JSF content, in conjunction with the actions that represent UI components.

FIXME - document current <output_body> requirement on included pages

JSF 8.2.3 Creating Components and Overriding Attributes

As component tags are encountered during the processing of a JSP page, the tag implementation must check the component tree for the existence of a corresponding UIComponent. Based on the results of this check:

- If no such UIComponent exists, a new component will be created and added as a child component of the component represented by the UIComponent custom action in which this action is nested (or the root component if this is the outermost component tag). All attributes specified on the component tag are used to initialize the attributes of the corresponding UIComponent instance.
- If such a UIComponent already exists, no new component is created. Instead, any attributes specified on the component tag are used to customize the attributes of the corresponding UIComponent, **unless** a value for that attribute has already been set (such as by the application handler in the *Invoke Application* phase).

Our example <input_text> tag supports a columns attribute, which allows the page author to configure the number of characters in the rendered input field, like this:

```
<h:input_text id="username"
    modelReference="logonBean.username" columns="32"/>
```

JSF 8.2.4 Representing Component Hierarchies

Nested structures of UIComponent custom actions will generally mirror the hierarchical relationships of the corresponding UIComponent instances in the component tree that is associated with each JSP page. For example, assume that a UIForm component (whose component id is *logonForm*) is nested inside a root UIPanel component. You might specify the contents of the form like this:

```
<h:form id="logonForm">
<table border="0">
<tr>
    <td><h:output_label id="usernameLabel"
```

```

        for="/logonForm/username">Username:
      </h:output_label></td>
      <td><h:input_text id="username"
        modelReference="logonBean.username"/></td>
    </tr>
    <tr>
      <td><h:output_label id="passwordLabel"
        for="/logonForm/password">Username:
      </h:output_label></td>
      <td><h:input_secret id="password"
        modelReference="logonBean.password"/></td>
    </tr>
    <tr>
      <td><h:command_button id="submitButton"
        type="SUBMIT" commandName="submit"/></td>
      <td> &nbsp; </td>
    </tr>
  </table>
</h:form>
```

Component tags with the `rendersChildren` property set to `true` will cause the child components in the component tree to be rendered along with the parent component. Therefore, the nested child components may, but need not, appear in the JSP page as tags.

JSF.8.2.5 Registering Event Listeners and Validators

Each JSF implementation is required to provide the core tag library (see Section JSF.8.5 “JSF Core Tag Library”), which includes tags that (when executed) create instances of a specified `ValueChangeListener`, `Actionlistener` or `Validator` implementation class, and register the created instance with the `UIComponent` associated with our most immediately surrounding `UIComponent` custom action. In addition, it is possible to register arbitrary attribute values (which are commonly used to configure converters and validators) on the component itself. *FIXME: add attribute description text.*

Using these facilities, the page author can manage all aspects of creating and configuring values associated with the component tree, without having to resort to Java code. For example:

```
<h:input_text id="username" modelReference="logonBean.username">
```

```

<f:validate_required/>
<f:validate_length minimum="6" />
</h:input_text>

```

associates two validators (a check for required input, and a minimum length check) with the username UIInput component being described.

Following are usage examples for the valuechanged_listener and action_listener tags.

```

<h:input_text>
    <f:valuechanged_listener
        type="custom.MyValueChangeListener" />
</h:input_text>
<h:command_button label="Login" commandName="login">
    <f:action_listener type="custom.MyActionListener" />
</h:command_button>

```

Each example causes a listener of the user specified type to be instantiated and added as a ValueChangedListener or ActionListener (respectively) to the enclosing UIComponent. If the user specified type is does not implement the proper listener interface a JSPEException must be thrown.

JSF 8.2.6 Using Facets

A *Facet* is a JavaServer Faces concept that denotes a UIComponent that has a special relationship to its parent UIComponent, as described in Section JSF 3.1.6 “Facet Management”. For example, imagine a table component that has special column headers and footers. Each header or footer can be denoted as a facet of the table component. It is useful to mark headers and footers as facets since there is a fixed set of these in the table, while there is a variable number of rows. Furthermore, it may be desirable to apply certain style attributes to headers and footers, while the regular row children of the table have a different style. In general, you can use Facets any time your component has a need to contain two different classes of sub-components: 1.) components belonging to a fixed set of things, (such as header or footer), and 2.) components not belonging to that set.

Facets can be used from JSP using the `<f:facet>` tag. Each Facet tag must have one and only one child tag, which must be a faces tag mapping to a UIComponent. For example:

```

<h:panel_list>
    <f:facet name="header">
        <h:output_text value="Account Number" />
    </f:facet>

```

```

</h:panel_list>

or

<h:panel_grid>
    <f:facet name="footer">
        <h:panel_group>
            <h:output_text value="Gross Margins"/>
            <h:output_text value="Goodwill"/>
            <h:output_text value="Gross Profit"/>
        </h:panel_group>
    </f:facet>
</h:panel_grid>
```

Note that it is permissible to have a single `<h:panel_group>` tag as the sole child of the `<f:facet>` tag.

JSE.8.2.7 Interoperability with Other Custom Action Libraries

It is permissible to use other custom action libraries, such as the JSP Standard Tag Library (JSTL) in the same JSP page with UIComponent custom actions that correspond to JSF components. When JSF component actions are nested inside tags from other libraries, the following behaviors must be supported:

- All JSF tags nested inside of, and positioned as siblings to the custom tags, must have a manually defined *component identifier*.
- Conditional custom actions (such as the `<c:if>` and `<c:choose>` actions in the JSP Standard Tag Library) may dynamically determine whether nested body content inside these tags is rendered or not. JSF component custom actions nested inside such tags will not be invoked if the outer tag chooses to skip its body. Therefore, no components will be created (in the component tree) to correspond to these custom actions. However, any such components created during a request processing lifecycle phase prior to *Render Response* must be included in the saved state information that will be communicated to the *Reconstitute Component Tree* phase of the subsequent request.
- Iterative custom actions (such as the `<c:forEach>` and `<c:forTokens>` actions in the JSP Standard Tag Library) may dynamically choose to process their nested body content zero or more times. JSF component custom actions nested within the body of such a tag must operate as follows:
 - If the body of the iterative custom action is never executed, nested JSF component custom actions will not be invoked. Therefore, no components will be created (in the component tree) to correspond to these custom actions. However, any such components created during a request processing lifecycle

phase prior to *Render Response* must be included in the saved state information that will be communicated to the *Reconstitute Component Tree* phase of the subsequent request.

- If the body of the iterative custom action is executed exactly once, nested JSF component custom actions must be treated exactly as if they were nested inside a conditional custom action (see the previous paragraph) that chose to render its nested body content.
- If the body of the iterative custom action is executed more than once, [FIXME - need a mechanism to have a `UIComponent` child be an **array** of `UIComponents` (one per iteration) as well as supporting model reference expressions that can use the loop index from the `LoopTagStatus` instance exported by a JSTL iteration tag.]

In addition to the general interoperability requirements described above, the following additional requirements must be satisfied when interoperating with tags from the JSP Standard Tag Library (JSTL):

- [FIXME - interaction with any `LocalizationContext` of tags from the “I18n-capable formatting tag library” described in Chapter 8 of the JSTL specification]
- [FIXME - other specific interoperability requirements to be determined]

[FIXME - we will probably need to specify that JSF depends on the availability of JSTL APIs so we can deal with things like
`javax.servlet.jsp.jstl.core.LoopTagStatus` and
`javax.servlet.jsp.jstl.fmt.LocalizationContext`].

JSF 8.2.8 Composing Pages from Multiple Sources

JSP pages can be composed from multiple sources using several mechanisms:

- The `<%@include%>` directive performs a compile-time inclusion of a specified source file into the page being compiled. From the perspective of JSF, such inclusions are transparent -- the page is compiled as if the inclusions had been performed before compilation was initiated.
- The `<jsp:include>` standard action performs a runtime dynamic inclusion of the results of including the response content of the requested page resource in place of the include action. Any JSF components created by execution of JSF component tags in the included page will be grafted onto the component tree, just as if the source text of the included page had appeared in the calling page at the position of the include action. [FIXME - do we need to restrict this to `flush="false"`?]
- Other mechanisms that perform a `RequestDispatcher.include()` on a JSP page from the same web application (such as use of the `include()` method on the `PageContext` object associated with a page, or accessing an internal resource

with the `<c:import>` tag of the JSP Standard Tag Library) must cause the component tree to be manipulated in the same manner as that described for the `<jsp:include>` standard action

- For mechanisms that aggregate content by other means (such as use of an `HttpURLConnection`, a `RequestDispatcher.include()` on a resource from a different web application acquired via `ServletContext.getServletContext()`, or accessing an external resource with the `<c:import>` tag of the JSP Standard Tag Library), only the response content of the aggregation request is available. Therefore, any use of JSF components in the generation of such a response are not combined with the component tree for the current page.

JSF.8.3 The `<f:use_faces/>` Tag

All of the JSF component tags used on a given page must be nested inside a `UseFaces` tag, whose implementation will be responsible for saving the state of the component tree during the execution of its `doEndTag()` method. This tag must be defined in the tag library descriptor of the JSF Core Tag Library described in Section JSF.8.5 “JSF Core Tag Library”.

JSF.8.4 UIComponent Custom Action Implementation Requirements

The custom action implementation classes for `UIComponent` custom actions must conform to all of the requirements defined in the JavaServer Pages Specification. In addition, they must meet the following JSF-specific requirements [FIXME - make more precise]

- Extend the `FacesTag` or `FacesBodyTag` base class, so that JSF implementations can recognize `UIComponent` custom actions versus others.
- Usage and interpretation of the `id` attribute
- Rules about overrides being applied only if the corresponding attribute was given a value in the page; otherwise, defaults from the component tree apply
- Other behavioral requirements specific to JSF?

JSF.8.5 JSF Core Tag Library

All JSF implementations must provide a tag library containing core actions (described below) that are independent of a particular RenderKit. The corresponding tag library descriptor must meet the following requirements:

- Must declare a tag library version (`<tlib-version>`) value of “1.0”.
- Must declare a JSP version dependency (`<tlib-version>`) value of “1.2”.
- Must declare a URI (`<uri>`) value of “<http://java.sun.com/jsf/core>”.

- Must be included in the META-INF directory of a JAR file containing the corresponding implementation classes, suitable for inclusion with a web application, such that the tag library descriptor will be located automatically by the algorithm described in Section 7.3 of the JavaServer Pages Specification (version 1.2).

JSP pages using the JSF standard tag library must declare it in a `taglib` like this (using the suggested prefix value):

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
```

Each action included in the Standard JSF Tag Library is documented in a subsection below, with the following outline for each action:

- **Name** -- The name of this tag, as used in a JSP page.
- **Short Description** -- A summary of the behavior implemented by this tag.
- **Syntax** -- One or more examples of using this tag, with the required and optional sets of attributes that may be used together.
- **Body Content** -- The type of nested content for this tag, using one of the standard values `empty`, `JSP`, or `tagdependent` as described in the JSP specification. This section also describes restrictions on the types of content (template text, JSF standard tags, JSF component tags, and/or other custom tags) that can be nested in the body of this tag.
- **Attributes** -- A table containing one row for each defined attribute for this tag. The following columns provide descriptive information about each attribute:
 - *Name* -- Name of this attribute, as it must be used in the page.
 - *Dyn* -- Boolean value indicating whether this attribute accepts dynamic values (documented in the `<rtexprvalue>` element).
 - *Type* -- Fully qualified Java class or primitive type of this attribute.
 - *Description* -- The functional meaning of this attribute's value.
- **Constraints** -- Additional constraints enforced by this action, such as combinations of attributes that may be used together.
- **Description** -- Details about the functionality provided by this action.

JSF 8.5.1 <f:action_listener>

Register an ActionListener instance for a UIComponent tag in which we are nested.

Syntax:

```
<f:action_listener type="fully-qualified-classname" />
```

Body Content:

empty.

Attributes:

Name	Dyn	Type	Description
type	false	String	Fully qualified Java class name of an ActionListener to be created and registered

Constraints:

- Must be nested inside a UIComponent tag.
- The UIComponent implementation class must define a public `addActionListener()` method that accepts an `ActionListener` parameter.
- The specified listener class must implement `javax.faces.event.ActionListener`.

Description:

Locates the UIComponent for the closest parent tag whose implementation class extends `javax.faces.webapp.FacesTag`. If the corresponding UIComponent instance was created by this component tag, instantiate an instance of the specified class, and register it by calling `addActionListener()`.

JSF 8.5.2 <f:attribute>

Add a String-valued attribute for a `UIComponent` tag in which we are nested.

Syntax:

```
<f:attribute name="attribute-name" value="attribute-value"/>
```

Body Content:

empty.

Attributes:

Name	Dyn	Type	Description
name	false	String	Name of the component attribute to be set
value	false	String	Value of the component attribute to be set

Constraints:

- Must be nested inside a `UIComponent` tag.

Description:

Locates the `UIComponent` for the closest parent tag whose implementation class extends `javax.faces.webapp.FacesTag`. If this component does not already have a component attribute with a name specified by this tag's `name` attribute, create a String valued component attribute with the name and value specified by this tag's attributes.

JSF 8.5.3 <f:facet>

Add a named facet (see Section JSF 3.1.6 “Facet Management”) for a `UIComponent` tag in which we are nested.

Syntax:

```
<f:facet name="facet-name" />
```

Body Content:

JSP.

Attributes:

Name	Dyn	Type	Description
name	false	String	Name of the facet to be created

Constraints:

- Must be nested inside a `UIComponent` tag.
- Exactly one `UIComponent` tag must be nested inside this tag (although the nested component tag could itself have nested children).

Description:

Locates the `UIComponent` for the closest parent tag whose implementation class extends `javax.faces.webapp.FacesTag`. If this component does not already have a facet with a name specified by this tag’s `name` attribute, create a facet with this name from the `UIComponent` tag that is nested within this tag.

JSF 8.5.4 <f:parameter>

Specify an optionally named UIPparameter component for a parent component.

Syntax:

Syntax 1: Unnamed direct value

```
<f:parameter id="componentId" value="parameter-value" />
```

Syntax 2: Unnamed indirect value

```
<f:parameter id="componentId"
    modelReference="model-reference-expression" />
```

Syntax 3: Named direct value

```
<f:parameter id="componentId" name="parameter-name"
    value="parameter-value" />
```

Syntax 4: Named indirect value

```
<f:parameter id="componentId" name="parameter-name"
    modelReference="model-reference-expression" />
```

Body Content:

empty.

Attributes:

Name	Dyn	Type	Description
id	false	String	Component identifier of a UIPparameter component
modelReference	false	String	Model reference expression to acquire the value of this parameter
name	false	String	Name of the parameter to be set
value	false	String	Value of the parameter to be set

Constraints:

- Must be nested inside a UIComponent tag.
- Must specify exactly one of the modelReference or value attributes.

Description:

Implements the standard FacesTag functionality to create a new UIParameter component, if one does not already exist in the component tree. Such components are used to dynamically configure parameter values for component tags in which it this tag is nested, such as substitution values for MessageFormat replacement patterns (`{0}`) in the `output_message` tag of the Standard HTML Tag Library.

The implementation class for this action must meet the following requirements:

- Must extend `javax.faces.FacesTag`.
- The `createComponent()` method must create and return a new UIParameter instance.
- The `getRendererType()` method must return `null`.

JSF 8.5.5 <f:use_faces>

Container tag for all JSF core and component tags used on a page.

Syntax:

```
<f:use_faces>  
    Nested template text and tags  
</f:use_faces>
```

Body Content:

JSP. May contain any combination of template text, other JSF tags, and tags from other custom tag libraries.

Attributes:

None.

Constraints:

- Any JSP-created response using actions from the JSF Core Tag Library, as well as actions extending javax.faces.webapp.FacesTag from other tag libraries, must be nest their use of such tags inside an occurrence of the <f:use_faces> action.
- JSP page fragments included via the standard <%@ include %> directive need not have their JSF actions embedded in a <f:use_faces> action, because the included template text and tags will be processed as part of the outer page as it is compiled, and the <f:use_faces> action on the outer page will meet the nesting requirement.
- JSP pages included via RequestDispatcher.include() or <jsp:include> may use an <f:use_faces>, but they need not do so if the calling page already contains such an action.

Description:

Provides the JSF implementation a convient place to perform state saving during the *Render Response* phase of the request processing lifecycle, if the implementation elects to save state as part of the response. The actual processing performed by this action is determined by the JSF implementation.

JSF 8.5.6 <f:validate_doublerange>

Register a `javax.faces.validator.DoubleRangeValidator` instance on our surrounding component

Syntax:

Syntax 1: Maximum only specified

```
<f:validate_doublerange maximum="543.21" />
```

Syntax 2: Minimum only specified

```
<f:validate_doublerange minimum="123.45" />
```

Syntax 3: Both maximum and minimum are specified

```
<f:validate_doublerange maximum="543.21" minimum="123.45" />
```

Body Content:

empty.

Attributes:

Name	Dyn	Type	Description
maximum	false	double	Maximum value allowed for this component
minimum	false	double	Minimum value allowed for this component

Constraints:

- Must be nested inside a `UIComponent` tag whose value is (or is convertible to) a double.
- Must specify either the `maximum` attribute, the `minimum` attribute, or both.
- If both limits are specified, the maximum limit must be greater than the minimum limit.

Description:

Creates, configures, and registers an instance of `javax.faces.validator.DoubleRangeValidator` on the `UIComponent` created by the `UIComponent` action within which we are nested.

The implementation class for this action must meet the following requirements:

- Must extend `javax.faces.ValidatorTag`.
- The `createValidator()` method must create, configure, and return a new `DoubleRangeValidator` instance.

JSF 8.5.7 <f:validate_length>

Register a javax.faces.validator.LengthValidator instance on our surrounding component

Syntax:

Syntax 1: Maximum only specified

```
<f:validate_length maximum="16" />
```

Syntax 2: Minimum only specified

```
<f:validate_length minimum="3" />
```

Syntax 3: Both maximum and minimum are specified

```
<f:validate_length maximum="16" minimum="3" />
```

Body Content:

empty.

Attributes:

Name	Dyn	Type	Description
maximum	false	int	Maximum length allowed for this component
minimum	false	int	Minimum length allowed for this component

Constraints:

- Must be nested inside a UIComponent tag whose value is a String.
- Must specify either the `maximum` attribute, the `minimum` attribute, or both.
- If both limits are specified, the `maximum` limit must be greater than the `minimum` limit.

Description:

Creates, configures, and registers an instance of `javax.faces.validator.LengthValidator` on the `UIComponent` created by the `UIComponent` action within which we are nested.

The implementation class for this action must meet the following requirements:

- Must extend `javax.faces.ValidatorTag`.
- The `createValidator()` method must create, configure, and return a new `LengthValidator` instance.

JSF 8.5.8 <f:validate_longrange>

Register a `javax.faces.validator.LongRangeValidator` instance on our surrounding component

Syntax:

Syntax 1: Maximum only specified

```
<f:validate_longrange maximum="543" />
```

Syntax 2: Minimum only specified

```
<f:validate_longrange minimum="123" />
```

Syntax 3: Both maximum and minimum are specified

```
<f:validate_longrange maximum="543" minimum="123" />
```

Body Content:

empty.

Attributes:

Name	Dyn	Type	Description
maximum	false	long	Maximum value allowed for this component
minimum	false	long	Minimum value allowed for this component

Constraints:

- Must be nested inside a `UIComponent` tag whose value is (or is convertible to) a long.
- Must specify either the `maximum` attribute, the `minimum` attribute, or both.
- If both limits are specified, the maximum limit must be greater than the minimum limit.

Description:

Creates, configures, and registers an instance of `javax.faces.validator.LongRangeValidator` on the `UIComponent` created by the `UIComponent` action within which we are nested.

The implementation class for this action must meet the following requirements:

- Must extend `javax.faces.ValidatorTag`.
- The `createValidator()` method must create, configure, and return a new `LongRangeValidator` instance.

JSF 8.5.9 <f:validate_required>

Register a javax.faces.validator.RequiredValidator instance on our surrounding component

Syntax:

```
<f:validate_required/>
```

Body Content:

empty.

Attributes:

None.

Constraints:

- Must be nested inside a UIComponent tag.

Description:

Creates, configures, and registers an instance of javax.faces.validator.RequiredValidator on the UIComponent created by the UIComponent action within which we are nested.

The implementation class for this action must meet the following requirements:

- Must extend javax.faces.ValidatorTag.
- The `createValidator()` method must create, configure, and return a new RequiredValidator instance.

JSF 8.5.10 <f:validate_stringrange>

Register a `javax.faces.validator.StringRangeValidator` instance on our surrounding component

Syntax:

Syntax 1: Maximum only specified

```
<f:validate_stringrange maximum="XYZ" />
```

Syntax 2: Minimum only specified

```
<f:validate_stringrange minimum="ABC" />
```

Syntax 3: Both maximum and minimum are specified

```
<f:validate_stringrange maximum="XYZ" minimum="ABC" />
```

Body Content:

empty.

Attributes:

Name	Dyn	Type	Description
maximum	false	String	Maximum value allowed for this component
minimum	false	String	Minimum value allowed for this component

Constraints:

- Must be nested inside a `UIComponent` tag whose value is (or is convertible to) a String.
- Must specify either the `maximum` attribute, the `minimum` attribute, or both.
- If both limits are specified, the `maximum` limit must be greater than the `minimum` limit.

Description:

Creates, configures, and registers an instance of `javax.faces.validator.StringValidator` on the `UIComponent` created by the `UIComponent` action within which we are nested.

The implementation class for this action must meet the following requirements:

- Must extend `javax.faces.ValidatorTag`.
- The `createValidator()` method must create, configure, and return a new `StringRangeValidator` instance.

JSF 8.5.11 <f:validator>

Create and register a Validator for a `UIComponent` tag in which we are nested.

Syntax:

```
<f:validator type="classname" />
```

Body Content:

empty.

Attributes:

Name	Dyn	Type	Description
type	false	String	Fully qualified class name of a class that extends <code>javax.faces.validator.Validator</code> .

Constraints:

- Must be nested inside a `UIComponent` tag.

Description:

Locates the `UIComponent` for the closest parent tag whose implementation class extends `javax.faces.webapp.FacesTag`. If this component was created by this execution of the surrounding tag, creates a new instance of a `Validator` subclass whose fully qualified class name is specified by the `type` attribute, and registers this instance with the component by calling `addValidator()`. The specified class must have a public zero-arguments constructor.

JSF 8.5.12 <f:valuechanged_listener>

Register an ValueChangeListener instance for a UIComponent tag in which we are nested.

Syntax:

```
<f:valuechanged_listener type="fully-qualified-classname" />
```

Body Content:

empty.

Attributes:

Name	Dyn	Type	Description
type	false	String	Fully qualified Java class name of an ValueChangeListener to be created and registered

Constraints:

- Must be nested inside a UIComponent tag.
- The UIComponent implementation class must define a public `addValueChangeListener()` method that accepts a `ValueChangeListener` parameter.
- The specified listener class must implement `javax.faces.event.ValueChangeListener`.

Description:

Locates the UIComponent for the closest parent tag whose implementation class extends `javax.faces.webapp.FacesTag`. If the corresponding UIComponent instance was created by this component tag, instantiate an instance of the specified class, and register it by calling `addValueChangeListener()`.

JSF.8.6 Standard HTML RenderKit Tag Library

All JSF implementations must provide a tag library containing actions (described below) that correspond to each valid combination of a supported component class (see Chapter 4) and a Renderer from the Standard HTML RenderKit (see Section JSF.7.4 “Standard HTML RenderKit Implementation”) that supports that component type. The tag library descriptor for this tag library must meet the following requirements:

- Must declare a tag library version (<tlib-version>) value of “**1.0**”.
- Must declare a JSP version dependency (<tlib-version>) value of “**1.2**”.
- Must declare a URI (<uri>) value of “**http://java.sun.com/jsf/html**”.
- Must be included in the META-INF directory of a JAR file containing the corresponding implementation classes, suitable for inclusion with a web application, such that the tag library descriptor will be located automatically by the algorithm described in Section 7.3 of the JavaServer Pages Specification (version 1.2).

JSP pages using the JSF standard tag library must declare it in a taglib like this (using the suggested prefix value):

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
```

Each action included in the Standard JSF Tag Library is documented in a subsection below, with the following outline for each action:

FIXME -- specify the mapping of component+rendererType to tags for all the combinations listed in Section JSF.7.4 “Standard HTML RenderKit Implementation”.

JSF.9

Using JSF In Web Applications

This specification provides JSF implementors significant freedom to differentiate themselves through innovative implementation techniques, as well as value-added features. However, to ensure that web applications based on JSF can be executed unchanged across different JSF implementations, the following additional requirements, defining how a JSF-based web application is assembled and configured, must be supported by all JSF implementations.

JSF.9.1 Web Application Deployment Descriptor

JSF-based applications are *web applications* that conform to the requirements of the Servlet Specification (version 2.3 or later), and also use the facilities defined in this specification. Conformant web applications are packaged in a *web application archive* (WAR), with a well-defined internal directory structure. A key element of a WAR is the *web application deployment descriptor*, an XML document that describes the configuration of the resources in this web application. This document is included in the WAR file itself, at resource path “/WEB-INF/web.xml”.

Portable JSF-based web applications must include the following configuration elements, in the appropriate portions of the web application deployment descriptor. Element values that are rendered in *italics* represent values that the application developer is free to choose. Element values rendered in **bold** represent values that must be utilized exactly as shown.

Executing the request processing lifecycle via other mechanisms is also allowed (for example, an MVC-based application framework can incorporate calling the correct Phase implementations in the correct order); however, all JSF implementations must support the functionality described in this chapter to ensure application portability.

JSF9.1.1 Servlet Definition

JSF implementations must provide request processing lifecycle services through a standard servlet, defined by this specification. This servlet must be defined, in the deployment descriptor, as follows:

```
<servlet>
    <servlet-name> faces-servlet-name </servlet-name>
    <servlet-class>
        javax.faces.webapp.FacesServlet
    </servlet-class>
</servlet>
```

The servlet name, denoted as *faces-servlet-name* above, may be any desired value; however, the same value must be used in the Servlet Mapping (see the next section).

JSF9.1.2 Servlet Mapping

All requests to a web application are mapped to a particular servlet based on matching a URL pattern (as defined in the Servlet Specification) against the portion of the request URL after the context path that selected this web application. The following mapping for the standard servlet providing portable JSF lifecycle processing must be supported:

```
<servlet-mapping>
    <servlet-name> faces-servlet-name </servlet-name>
    <url-pattern>
        /faces/*
    </url-pattern>
</servlet-mapping>
```

The servlet name, denoted by *faces-servlet-name* above, may be any desired value; however, the same value must be used in the Servlet Definition (see the previous section).

JSF9.1.3 Application Configuration Parameters

Servlet containers support application configuration parameters that may be customized by including `<context-param>` elements in the web application deployment descriptor. All JSF implementations are required to support the following application configuration parameter names:

- `javax.faces.lifecycle.LIFECYCLE_ID` - Lifecycle identifier of the Lifecycle instance to be used when processing JSF requests in this web application. If not specified, the JSF default instance, identified by `LifecycleFactory.DEFAULT_LIFECYCLE`, will be used.

FIXME - list of other standard `javax.faces.xxxxx` parameters to be supported

JSF implementations may choose to support additional configuration parameters, as well as additional mechanisms to customize the JSF implementation; however, applications that rely on these facilities will not be portable to other JSF implementations.

JSF.9.2 Included Classes and Resources

A JSF-based application will rely on a combination of APIs, and corresponding implementation classes and resources, in addition to its own classes and resources. The web application archive structure identifies two standard locations for classes and resources that will be automatically made available when a web application is deployed:

- `/WEB-INF/classes` - A directory containing unpacked class and resource files.
- `/WEB-INF/lib` - A directory containing JAR files that themselves contain class files and resources.

In addition, servlet containers typically provide mechanisms to share classes and resources across one or more web applications, without requiring them to be included inside the web application archive itself.

The following sections describe how various subsets of the required classes and resources should be packaged, and how they should be made available.

JSF.9.2.1 Application-Specific Classes and Resources

Application-specific classes and resources should be included in `/WEB-INF/classes` or `/WEB-INF/lib`, so that they are automatically made available upon application deployment.

JSF.9.2.2 Servlet and JSP API Classes (`javax.servlet.*`)

These classes will typically be made available to all web applications using the shared classes facilities of the servlet container¹. Therefore, these classes should not be included inside the web application archive.

1. This is already a requirement for all J2EE containers.

JSF.9.2.3 JavaServer Faces API Classes (`javax.faces.*`)

These classes describe the fundamental APIs provided by all JSF implementations. They are generally packaged in a JAR file named `jsf-api.jar` (although this name is not required). The JSF API classes should be installed using the shared classes facility of your servlet container; however, they may also be included (in the `/WEB-INF/lib` directory).

At some future time, JavaServer Faces might become part of the Java2 Enterprise Edition (J2EE) platform, at which time the container will be required to provide these classes through a shared class facility.

JSF.9.2.4 JavaServer Faces Implementation Classes

These classes and resources comprise the implementation of the JSF APIs that is provided by a *JSF Implementor*. Typically, such classes will be provided in the form of one or more JAR files, which can be either installed with the container's shared class facility, or included in the `/WEB-INF/lib` directory of a web application archive.

JSF.9.2.4.1 FactoryFinder

The `javax.faces.FactoryFinder` class implements a standard discovery mechanism for locating JSF-implementation-specific implementations of several factory classes defined by JSF. The discovery method is static, so that it may be conveniently called from JSF implementation or application code, without requiring a reference to some global container.

- `public static Object getFactory(String factoryName);`

Create (if necessary) and return an instance of the implementation class for the specified factory name. This method must ensure that there is exactly one instance for each specified factory name, per web application supported in a servlet container. For a given factory name, the following process is used to identify the name of the factory implementation class to be instantiated:

- If there is a system property whose name matches the requested factory name, its value is assumed to be the name of the factory implementation class to use.
- If there is a `faces.properties` resource file visible to the web application class loader of the calling application, and if this resource file defines a property whose name matches the requested factory name, its value is assumed to be the name of the factory implementation class to use.
- If a `META-INF/services/{factory-name}` resource file is visible to the web application class loader of the calling application (typically as a result of it being included in a JAR file containing the corresponding implementation class), the first line of this resource file is assumed to contain the name of the factory implementation class to use.
- If none of the above conditions are satisfied, a `FacesException` is thrown.

Once the name of the factory implementation class to use is identified, FactoryFinder must load this class from the web application class loader, and then instantiate an instance for the current web application. Any subsequent request for the same factory name, from the same web application, must cause the previously created instance to be returned.

JSF implementations must also include implementations of the several factory classes. In order to be dynamically instantiated according to the algorithm defined above, the factory implementation class must include a public, no-arguments constructor. Factory class implementations must be provided for the following factory names:

- `javax.faces.context.ConverterFactory` (`FactoryFinder.CONVERTER_FACTORY`) - Factory for Converter instances.
- `javax.faces.context.FacesContextFactory` (`FactoryFinder.FACES_CONTEXT_FACTORY`) - Factory for FacesContext instances.
- `javax.faces.context.LifecycleFactory` (`FactoryFinder.LIFECYCLE_FACTORY`) - Factory for Lifecycle instances.
- `javax.faces.context.MessageResourcesFactory` (`FactoryFinder.MESSAGE_RESOURCES_FACTORY`) - Factory for MessageResources instances.
- `javax.faces.context.RenderKitFactory` (`FactoryFinder.RENDER_KIT_FACTORY`) - Factory for RenderKit instances.
- `javax.faces.context.TreeFactory` (`FactoryFinder.TREE_FACTORY`) - Factory for Tree instances.

JSF.9.2.4.2FacesServlet

FacesServlet is an implementation of `javax.servlet.Servlet` that accepts incoming requests and passes them to the appropriate Lifecycle implementation for processing. This servlet must be declared in the web application deployment descriptor, as described in Section JSF.9.1.1 “Servlet Definition”, and mapped to a standard URL pattern as described in Section JSF.9.1.2 “Servlet Mapping”.

- `public void init(ServletConfig config);`

Acquire and store references to the FacesContextFactory and LifecycleFactory instances to be used in this web application.

- `public void destroy();`

Release the FacesContextFactory and LifecycleFactory references that were acquired during execution of the `init()` method.

- `public void service(ServletRequest request, ServletResponse response);`

For each incoming request, the following processing is performed:

- Using the `FacesContextFactory` instance stored during the `init()` method, call the `createFacesContext()` method to acquire a `FacesContext` instance with which to process the current request.
- Store the `FacesContext` instance as a request attribute under key “`javax.faces.context.FacesContext`” (`FacesContext.FACES_CONTEXT_ATTR`).
- Using the `LifecycleFactory` instance stored during the `init()` method, call the `createLifecycle()` method to acquire a `Lifecycle` instance with which to perform the request processing lifecycle for the current request.
- Call the `execute()` method of the `Lifecycle` instance, passing the `FacesContext` instance for this request as a parameter. If the `execute()` method throws a `FacesException`, rethrow it as a `ServletException` with the `FacesException` as the root cause.
- Remove the request attribute containing the `FacesContext` instance.
- Call the `release()` method on the `FacesContext` instance, allowing it to be returned to a pool if the JSF implementation uses one.

JSF.9.2.4.3FacesTag

`FacesTag` is a subclass of `javax.servlet.jsp.tagext.TagSupport`, and must be the base class for any JSP custom tag that corresponds to a JSF `UIComponent`. It supports all of the standard functionality of `TagSupport`, plus the following functionality that is specific to JSF:

- `protected UIComponent findComponent();`

Using the current value of the `id` property of this tag handler, locate and return the corresponding `UIComponent` instance from the component tree, in accordance with the rules outlined in Section JSF.8.2.2 “Defining Components in a Page”, above.

If your tag needs to implement `javax.servlet.jsp.tagext.BodyTag` support, it should subclass `FacesBodyTag` (described in the following section) instead.

JSF.9.2.4.4FacesBodyTag

`FacesBodyTag` is a subclass of `FacesTag`, so it inherits all of the JSF-specific functionality described in the preceding section. In addition, this class implements the standard functionality provided by `javax.servlet.jsp.BodyTagSupport`, so it is useful as the base class for JSF custom action implementations that must process their body content.

JSF.9.3 Included Configuration Files

JSF defines portable configuration file formats (as XML documents) for standard configuration information such as message catalogs ... **FIXME** - document the formats and default locations here, with references back to the configuration parameters that support customization of these locations.

JSF.10

JavaServer Faces API

This chapter contains the Java API documentation for all interfaces and classes that are part of the JavaServer Faces Specification.

JSF.10.1 Package javax.faces

Package javax.faces

Class Summary

Classes

`FactoryFinder`_o

FactoryFinder implements the standard discovery algorithm for all factory objects specified in the JavaServer Faces APIs.

Exceptions

`FacesException`_o

This class encapsulates general JavaServer Faces exceptions.

javax.faces FacesException

Declaration

```
public class FacesException extends java.lang.RuntimeException  
  
java.lang.Object  
|  
+--java.lang.Throwable  
|  
+--java.lang.Exception  
|  
+--java.lang.RuntimeException  
|  
+--javax.faces.FacesException
```

All Implemented Interfaces: `java.io.Serializable`

Direct Known Subclasses: `javax.faces.event.AbortProcessingException`,
`javax.faces.convert.ConverterException`,

Description

This class encapsulates general JavaServer Faces exceptions.

Member Summary

Constructors

```
FacesException()  
    Construct a new exception with no detail message or root cause.  
FacesException(java.lang.String message)  
    Construct a new exception with the specified detail message and no root cause.  
FacesException(java.lang.String message, java.lang.Throwable cause)  
    Construct a new exception with the specified detail message and root cause.  
FacesException(java.lang.Throwable cause)  
    Construct a new exception with the specified root cause.
```

Methods

```
java.lang.Throwable getCause()  
    Return the cause of this exception, or null if the cause is nonexistent or unknown.
```

Inherited Member Summary

Methods inherited from class `Object`

`clone()`, `equals(Object)`, `finalize()`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`,
`wait()`, `wait()`, `wait()`

Methods inherited from class `Throwable`

`fillInStackTrace()`, `getLocalizedMessage()`, `getMessage()`,
`printStackTrace(PrintWriter)`, `printStackTrace(PrintWriter)`,
`printStackTrace(PrintWriter)`, `toString()`

Constructors

`FacesException()`

```
public FacesException()
```

Construct a new exception with no detail message or root cause.

`FacesException(String)`

```
public FacesException(java.lang.String message)
```

Construct a new exception with the specified detail message and no root cause.

Parameters:

`message` - The detail message for this exception

`FacesException(String, Throwable)`

```
public FacesException(java.lang.String message, java.lang.Throwable cause)
```

Construct a new exception with the specified detail message and root cause.

Parameters:

`message` - The detail message for this exception

`cause` - The root cause for this exception

`FacesException(Throwable)`

```
public FacesException(java.lang.Throwable cause)
```

Construct a new exception with the specified root cause. The detail message will be set to (`cause == null ? null : cause.toString()`)

Parameters:

`cause` - The root cause for this exception

Methods

getCause()

```
public java.lang.Throwable getCause()
```

Return the cause of this exception, or `null` if the cause is nonexistent or unknown.

javax.faces FactoryFinder

Declaration

```
public final class FactoryFinder  
  
java.lang.Object  
|  
+--javax.faces.FactoryFinder
```

Description

FactoryFinder implements the standard discovery algorithm for all factory objects specified in the JavaServer Faces APIs. For a given factory class name, a corresponding implementation class is searched for based on the following algorithm:

- If there is a system property whose name matches the name of the desired factory class, its value is assumed to be the name of the factory implementation class to use.
- If a `faces.properties` file is visible as a resource to the web application class loader for the calling application, and this properties file contains property whose name matches the name of the desired factory class, its value is assumed to be the name of the factory implementation class to use.
- If a `META-INF/services/{factory-class-name}` resource is visible to the web application class loader for the calling application (typically as a result of being present in the manifest of a JAR file), its first line is read and assumed to be the name of the factory implementation class to use.

Once the name of the factory implementation class is located, the web application class loader for the calling application is requested to load this class, and a corresponding instance of the class will be created. A side effect of this rule is that each web application will receive its own instance of each factory class, whether the JavaServer Faces implementation is included within the web application or is made visible through the container's facilities for shared libraries.

Factory implementation classes must include a zero-arguments constructor in order to be successfully instantiated.

Member Summary

Fields

static CONVENTER_FACTORY ₀	
java.lang.String	The property name for the <code>javax.faces.convert.ConverterFactory</code> class name.

Member Summary

static	FACES_CONTEXT_FACTORY ₀	
java.lang.String	The property name for the javax.faces.context.FacesContextFactory ₀ class name.	
static	LIFECYCLE_FACTORY ₀	
java.lang.String	The property name for the javax.faces.lifecycle.LifecycleFactory ₀ class name.	
static	MESSAGE_RESOURCES_FACTORY ₀	
java.lang.String	The property name for the javax.faces.context.MessageResourcesFactory ₀ class name.	
static	RENDER_KIT_FACTORY ₀	
java.lang.String	The property name for the javax.faces.render.RenderKitFactory ₀ class name.	
static	TREE_FACTORY ₀	
java.lang.String	The property name for the javax.faces.tree.TreeFactory ₀ class name.	

Methods

static	getFactory(java.lang.String factoryName) ₀	
java.lang.Object	Create (if necessary) and return a per-web-application instance of the appropriate implementation class for the specified JavaServer Faces factory class, based on the discovery algorithm described in the class description.	
static void	releaseFactories() ₀	Release any references to factory instances associated with the class loader for the calling web application.

Inherited Member Summary

Methods inherited from class Object

clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(),
toString(), wait(), wait(), wait()

Fields

CONVERTER_FACTORY

public static final java.lang.String CONVERTER_FACTORY

The property name for the javax.faces.convert.ConverterFactory₀ class name.

FACES_CONTEXT_FACTORY

public static final java.lang.String FACES_CONTEXT_FACTORY

The property name for the `javax.faces.context.FacesContextFactory`₀ class name.

LIFECYCLE_FACTORY

```
public static final java.lang.String LIFECYCLE_FACTORY
```

The property name for the `javax.faces.lifecycle.LifecycleFactory`₀ class name.

MESSAGE_RESOURCES_FACTORY

```
public static final java.lang.String MESSAGE_RESOURCES_FACTORY
```

The property name for the `javax.faces.context.MessageResourcesFactory`₀ class name.

RENDER_KIT_FACTORY

```
public static final java.lang.String RENDER_KIT_FACTORY
```

The property name for the `javax.faces.render.RenderKitFactory`₀ class name.

TREE_FACTORY

```
public static final java.lang.String TREE_FACTORY
```

The property name for the `javax.faces.tree.TreeFactory`₀ class name.

Methods

getFactory(String)

```
public static java.lang.Object getFactory(java.lang.String factoryName)  
    throws FacesException
```

Create (if necessary) and return a per-web-application instance of the appropriate implementation class for the specified JavaServer Faces factory class, based on the discovery algorithm described in the class description.

Parameters:

`factoryName` - Fully qualified name of the JavaServer Faces factory for which an implementation instance is requested

Throws:

`FacesException`₀ - if the web application class loader cannot be identified

`FacesException`₀ - if an instance of the configured factory implementation class cannot be loaded

`FacesException` - if an instance of the configured factory implementation class cannot be instantiated

`java.lang.IllegalArgumentException` - if `factoryName` does not identify a standard JavaServer Faces factory name

`java.lang.IllegalStateException` - if there is no configured factory implementation class for the specified factory name

`NullPointerException` - if `factoryname` is null

releaseFactories()

```
public static void releaseFactories()
    throws FacesException
```

Release any references to factory instances associated with the class loader for the calling web application. This method should be called as apart of web application shutdown in a container where the JavaServer Faces API classes are part of the container itself, rather than being included inside the web application.

Throws:

`FacesException` - if the web application class loader cannot be identified

JSF.10.2 Package `javax.faces.component`

Package `javax.faces.component`

Description

Fundamental APIs for user interface components, and concrete component implementation classes.

Class Summary

Interfaces

`NamingContainer`

NamingContainer is an interface that must be implemented by any `UIComponent`s that wants to be a naming container.

`UIComponent`

UIComponent is the base interface for all user interface components in JavaServer Faces.

Classes

`AttributeDescriptor`

An **AttributeDescriptor** describes a single characteristic of a `UIComponent`, that might be of interest to other classes, such as a `javax.faces.validator.Validator` or a `javax.faces.render.Renderer`.

`NamingContainerSupport`

NamingContainerSupport is a utility class that may be utilized by `UIComponent`s that implement `NamingContainer` to delegate naming container management methods.

`SelectItem`

SelectItem represents a single *item* in the list of supported *items* associated with a `UISelectMany` or `UISelectOne` component.

`UICommand`

UICommand is a `UIComponent` that represents a user interface component which, when activated by the user, triggers an application specific “command” or “action”.

`UIComponentBase`

UIComponentBase is a convenience base class that implements the default concrete behavior of all methods defined by `UIComponent`, except `getComponentType()`.

`UIForm`

UIForm is a `UIComponent` that represents an input form to be presented to the user, and whose child components represent (among other things) the input fields to be included when the form is submitted.

`UIGraphic`

UIGraphic is a `UIComponent` that displays a graphical image to the user.

`UIInput`

UIInput is a `UIComponent` that represents a component that both displays output to the user (like `UIOutput` components do) and includes request parameters on the subsequent request that need to be decoded.

`UINamingContainer`

UINamingContainer is a convenience base class for components that wish to implement `NamingContainer` functionality.

Class Summary	
UIOutput ₀	UIOutput is a <code>UIComponent₀</code> that displays output to the user.
UIPanel ₀	UIPanel is a <code>UIComponent₀</code> that manages the layout of its child components.
UIParameter ₀	UIParameter is a <code>UIComponent₀</code> that represents an optionally named configuration parameter for a parent component.
UISelectBoolean ₀	UISelectBoolean is a <code>UIComponent₀</code> that represents a single boolean (<code>true</code> or <code>false</code>) value.
UISelectItem ₀	UISelectItem is a component that may be nested inside a <code>UISelectMany₀</code> or <code>UISelectOne₀</code> component, and causes the addition of a <code>SelectItem₀</code> instance to the list of available options for the parent component.
UISelectItems ₀	UISelectItems is a component that may be nested inside a <code>UISelectMany₀</code> or <code>UISelectOne₀</code> component, and causes the addition of one or more <code>SelectItem₀</code> instances to the list of available options in the parent component.
UISelectMany ₀	UISelectMany is a <code>UIComponent₀</code> that represents the user's choice of a zero or more items from among a discrete set of available options.
UISelectOne ₀	UISelectOne is a <code>UIComponent₀</code> that represents the user's choice of a single item from among a discrete set of available options.

javax.faces.component AttributeDescriptor

Declaration

```
public abstract class AttributeDescriptor  
  
java.lang.Object  
|  
+--javax.faces.component.AttributeDescriptor
```

Description

An **AttributeDescriptor** describes a single characteristic of a *UIComponent*, that might be of interest to other classes, such as a *javax.faces.validator.Validator*, or a *javax.faces.render.Renderer*. This information is particularly useful in tools that wish to automate the creation of user interfaces based on JavaServer Faces components.

Member Summary

Constructors

```
AttributeDescriptor()
```

Methods

abstract	getDescription()	
java.lang.String		Return a brief description of this attribute, useful when rendering help text in a tool, localized for the default Locale for this instance of the Java Virtual Machine.
abstract	getDescription(java.util.Locale locale)	
java.lang.String		Return a brief description of this attribute, useful when rendering help text in a tool, localized for the specified Locale.
abstract	getDisplayName()	
java.lang.String		Return a short displayable name of this attribute, useful in constructing the user interface of a tool, localized for the default Locale for this instance of the Java Virtual Machine.
abstract	getDisplayName(java.util.Locale locale)	
java.lang.String		Return a short displayable name of this attribute, useful in constructing the user interface of a tool, localized for the specified Locale.
abstract	getName()	
java.lang.String		Return the attribute name of the attribute described by this AttributeDescriptor.
abstract	getType()	
java.lang.Class		Return the Java class representing the data type of the legal values for this attribute.

Inherited Member Summary

Methods inherited from class Object

`clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(),
toString(), wait(), wait(), wait()`

Constructors

AttributeDescriptor()

```
public AttributeDescriptor()
```

Methods

getDescription()

```
public abstract java.lang.String getDescription()
```

Return a brief description of this attribute, useful when rendering help text in a tool, localized for the default Locale for this instance of the Java Virtual Machine.

get_description(Locale)

```
public abstract java.lang.String getDescription(java.util.Locale locale)
```

Return a brief description of this attribute, useful when rendering help text in a tool, localized for the specified Locale.

Parameters:

`locale` - Locale for which to retrieve a localized description

Throws:

`NullPointerException` - if `locale` is null

getDisplayName()

```
public abstract java.lang.String getDisplayName()
```

Return a short displayable name of this attribute, useful in constructing the user interface of a tool, localized for the default Locale for this instance of the Java Virtual Machine.

getDisplayName(Locale)

```
public abstract java.lang.String getDisplayName(java.util.Locale locale)
```

Return a short displayable name of this attribute, useful in constructing the user interface of a tool, localized for the specified Locale.

Parameters:

locale - Locale for which to retrieve a display name

Throws:

NullPointerException - if locale is null

getName()

```
public abstract java.lang.String getName( )
```

Return the attribute name of the attribute described by this AttributeDescriptor.

getType()

```
public abstract java.lang.Class getType( )
```

Return the Java class representing the data type of the legal values for this attribute.

javax.faces.component NamingContainer

Declaration

```
public interface NamingContainer
```

All Known Implementing Classes: NamingContainerSupport₀,
UINamingContainer₀, UISelectBase

Description

NamingContainer is an interface that must be implemented by any **UIComponent₀** that wants to be a naming container.

Member Summary

Methods

void addComponentToNamespace(UIComponent component) ₀ Add the specified UIComponent₀ to the namespace of this naming container.	UIComponent findComponentInNamespace(java.lang.String name) ₀ Find and return a UIComponent₀ in this namespace, if it is present; otherwise return null.
java.lang.String generateClientId() ₀ Generate an identifier for a component, suitable for communication to a client.	void removeComponentFromNamespace(UIComponent component) ₀ Remove the specified UIComponent₀ from the namespace of this naming container, if it is present.

Methods

addComponentToNamespace(UIComponent)

```
public void addComponentToNamespace(javax.faces.component.UIComponent0 component)
```

Add the specified **UIComponent₀** to the namespace of this naming container.

Parameters:

component - The **UIComponent₀** to be added

Throws:

`java.lang.IllegalArgumentException` - if the specified component does not have a `componentId`
`java.lang.IllegalStateException` - if the component identifier of the specified component is not unique within the namespace of this naming container
`NullPointerException` - if `component` is `null`

findComponentInNamespace(String)

```
public javax.faces.component.UIComponent_0
    findComponentInNamespace(java.lang.String name)
```

Find and return a `UIComponent0` in this namespace, if it is present; otherwise return `null`.

If the argument name does not contain any `UIComponent.SEPARATOR_CHAR0` characters, it is interpreted to be a name in the namespace of this naming container.

If the argument name does contain `UIComponent.SEPARATOR_CHAR0` characters, each segment between `UIComponent.SEPARATOR_CHAR0` is treated as a component identifier in its own namespace, which are searched for in child naming containers of this naming container, from left to right.

Parameters:

`name` - Identifier of the desired component

Throws:

`java.lang.IllegalArgumentException` - if `name` is malformed
`NullPointerException` - if `name` is `null`

generateClientId()

```
public java.lang.String generateClientId()
```

Generate an identifier for a component, suitable for communication to a client. The identifier returned from this method must be unique within this namespace.

removeComponentFromNamespace(UIComponent)

```
public void removeComponentFromNamespace(javax.faces.component.UIComponent_0
    component)
```

Remove the specified `UIComponent0` from the namespace of this naming container, if it is present.

Parameters:

`component` - The `UIComponent0` to be removed

Throws:

`java.lang.IllegalArgumentException` - if the specified component does not have a `componentId`

`NullPointerException` - if `component` is `null`

javax.faces.component

NamingContainerSupport

Declaration

```
public class NamingContainerSupport implements NamingContainer0, java.io.Serializable  
java.lang.Object  
|  
+-- javax.faces.component.NamingContainerSupport
```

All Implemented Interfaces: NamingContainer₀, java.io.Serializable

Description

NamingContainerSupport is a utility class that may be utilized by UIComponent₀s that implement NamingContainer₀ to delegate naming container management methods.

Typical usage in a UIComponent₀ implementation class would be:

```
public class MyComponent extends UIComponentBase  
    implements NamingContainer {  
    private NamingContainerSupport ncs = new NamingContainerSupport();  
    ...  
    public void addComponentToNamespace(UIComponent component) {  
        ncs.addComponentToNamespace(component);  
    }  
    public void removeComponentFromNamespace(UIComponent component) {  
        ncs.removeComponentFromNamespace(component);  
    }  
    public UIComponent findComponentInNamespace(String name) {  
        return ncs.findComponentInNamespace(name);  
    }  
    public String generateClientId() {  
        return ncs.generateClientId();  
    }  
}
```

Member Summary

Constructors

`NamingContainerSupport()0`

Methods

```
void addComponentToNamespace(UIComponent component)0  
UIComponent findComponentInNamespace(java.lang.String name)0  
java.lang.String generateClientId()0
```

Member Summary

```
void removeComponentFromNamespace(UIComponent component)
```

Inherited Member Summary**Methods inherited from class Object**

```
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(),
toString(), wait(), wait(), wait()
```

Constructors

NamingContainerSupport()

```
public NamingContainerSupport()
```

Methods

addComponentToNamespace(UIComponent)

```
public void addComponentToNamespace(javax.faces.component.UIComponent component)
```

Description copied from interface: `javax.faces.component.NamingContainer`

Add the specified `UIComponent` to the namespace of this naming container.

Specified By: `addComponentToNamespace` in interface `NamingContainer`

findComponentInNamespace(String)

```
public javax.faces.component.UIComponent
    findComponentInNamespace(java.lang.String name)
```

Description copied from interface: `javax.faces.component.NamingContainer`

Find and return a `UIComponent` in this namespace, if it is present; otherwise return null.

If the argument name does not contain any `UIComponent.SEPARATOR_CHAR` characters, it is interpreted to be a name in the namespace of this naming container.

If the argument name does contain `UIComponent.SEPARATOR_CHAR` characters, each segment between `UIComponent.SEPARATOR_CHAR` is treated as a component identifier in its

own namespace, which are searched for in child naming containers of this naming container, from left to right.

Specified By: `findComponentInNamespace0` in interface `NamingContainer0`

generateClientId()

```
public java.lang.String generateClientId()
```

Description copied from interface: `javax.faces.component.NamingContainer0`

Generate an identifier for a component, suitable for communication to a client. The identifier returned from this method must be unique within this namespace.

Specified By: `generateClientId0` in interface `NamingContainer0`

removeComponentFromNamespace(UIComponent)

```
public void removeComponentFromNamespace( javax.faces.component.UIComponent0
                                         component )
```

Description copied from interface: `javax.faces.component.NamingContainer0`

Remove the specified `UIComponent0` from the namespace of this naming container, if it is present.

Specified By: `removeComponentFromNamespace0` in interface `NamingContainer0`

javax.faces.component SelectItem

Declaration

```
public class SelectItem implements java.io.Serializable  
  
java.lang.Object  
|  
+--javax.faces.component.SelectItem
```

All Implemented Interfaces: java.io.Serializable

Description

SelectItem represents a single *item* in the list of supported *items* associated with a **UISelectMany**₀ or **UISelectOne**₀ component. It is not itself a **UIComponent**₀, but one or more **SelectItem** instances can be utilized in a **UISelectItem**₀ component.

Member Summary

Constructors

```
SelectItem(java.lang.Object value, java.lang.String label,  
          java.lang.String description)0  
Construct a UISelectItem instance with the specified property values.
```

Methods

```
java.lang.String getDescription()0  
Return a description of this item, for use in development tools.  
java.lang.String getLabel()0  
Return the label of this item, to be rendered visibly for the user.  
java.lang.Object getValue()0  
Return the value of this item, to be returned as a request parameter if this item is  
selected by the user.
```

Inherited Member Summary

Methods inherited from class Object

Inherited Member Summary

```
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(),
toString(), wait(), wait(), wait()
```

Constructors

SelectItem(Object, String, String)

```
public SelectItem(java.lang.Object value, java.lang.String label,
                 java.lang.String description)
```

Construct a UISelectItem instance with the specified property values.

Parameters:

value - Value to be returned as a request parameter if this item is selected by the user

label - Label to be rendered for this item in the response

description - Description of this item, for use in tools

Methods

getDescription()

```
public java.lang.String getDescription()
```

Return a description of this item, for use in development tools.

getLabel()

```
public java.lang.String getLabel()
```

Return the label of this item, to be rendered visibly for the user.

getValue()

```
public java.lang.Object getValue()
```

Return the value of this item, to be returned as a request parameter if this item is selected by the user.

javax.faces.component UICommand

Declaration

```
public class UICommand extends UIComponentBase0  
  
java.lang.Object  
|  
+--javax.faces.component.UIComponentBase0  
|  
+--javax.faces.component.UICommand
```

All Implemented Interfaces: java.io.Serializable, UIComponent₀

Description

UICommand is a UIComponent₀, that represents a user interface component which, when activated by the user, triggers an application specific “command” or “action”. Such a component is typically rendered as a push button, a menu item, or a hyperlink.

When the decode() method of this UICommand₀, or its corresponding javax.faces.render.Renderer₀, detects that this control has been activated, it will queue an javax.faces.event.ActionEvent₀. Later on, the broadcast() method will ensure that this event is broadcast to all interested listeners.

Member Summary

Fields

protected listeners ₀ java.util.List[]	Array of java.util.List s of javax.faces.event.ActionListener ₀ s registered for particular phases.
static TYPE ₀ java.lang.String	The component type of this UIComponent ₀ subclass.

Constructors

UICommand()₀

Methods

void addActionListener(javax.faces.event.ActionListener listener) ₀	Add a new javax.faces.event.ActionListener ₀ to the set of listeners interested in being notified when javax.faces.event.ActionEvent ₀ s occur.
--	---

Member Summary

```
protected void broadcast(javax.faces.event.ActionEvent event, java.util.List<? extends javax.faces.event.ActionListener> list)
    Broadcast the specified javax.faces.event.ActionEvent, to the
    javax.faces.event.ActionListener's on the specified list (if any)

boolean broadcast(javax.faces.event.FacesEvent event,
    javax.faces.event.PhaseId phaseId)
    Broadcast the specified javax.faces.event.FacesEvent, to all registered
    event listeners who have expressed an interest in events of this type, for the specified
    javax.faces.event.PhaseId.

void decode(javax.faces.context.FacesContext context)
    Enqueue a javax.faces.event.FormEvent event to the application
    identifying the form submission that has occurred, along with the command name of
    the UICommand that caused the form to be submitted, if any.

void encodeEnd(javax.faces.context.FacesContext context)
void fireActionEvent(javax.faces.context.FacesContext context)
    Queue an javax.faces.event.ActionEvent for processing during the next
    event processing cycle.

java.lang.String getCommandName()
    Return the command name for this UICommand.

java.lang.String getComponentType()
void removeActionListener(javax.faces.event.ActionListener listener)
    Remove an existing javax.faces.event.ActionListener (if any) from
    the set of listeners interested in being notified when
    javax.faces.event.ActionEvent's occur.

void setCommandName(java.lang.String commandName)
    Set the command name for this UICommand.

void updateModel(javax.faces.context.FacesContext context)
    Override the default behavior and perform no model update.
```

Inherited Member Summary**Fields inherited from interface `UIComponent`**

`CLIENT_ID_ATTR`, `FACET_PARENT`, `FACET_PARENT_ATTR`, `SEPARATOR_CHAR`

Methods inherited from class `Object`

`clone()`, `equals(Object)`, `finalize()`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`,
`toString()`, `wait()`, `wait()`, `wait()`

Methods inherited from class `UIComponentBase`

Inherited Member Summary

```
addChild(int, UIComponent)o, addChild(int, UIComponent)o, addFacet(String, UIComponent)o, addValidator(Validator)o, clearChildren()o, clearFacets()o, clearValidators()o, containsChild(UIComponent)o, currentValue(FacesContext)o, encodeBegin(FacesContext)o, encodeChildren(FacesContext)o, findComponent(String)o, getAttribute(String)o, getAttributeNames()o, getChild(int)o, getChildCount()o, getChildren()o, getClientId(FacesContext)o, getComponentId()o, getConverter()o, getFacet(String)o, getFacetNames()o, getFacetsAndChildren()o, getModelReference()o, getParent()o, getRendererType()o, getRendersChildren()o, getRendersSelf()o, getValidators()o, getValue()o, isRendered()o, isValid()o, processDecodes(FacesContext)o, processUpdates(FacesContext)o, processValidators(FacesContext)o, removeChild(UIComponent)o, removeChild(UIComponent)o, removeFacet(String)o, removeValidator(Validator)o, setAttribute(String, Object)o, setComponentId(String)o, setConverter(String)o, setModelReference(String)o, setRendered(boolean)o, setRendererType(String)o, setValid(boolean)o, setValue(Object)o, validate(FacesContext)o
```

Fields

listeners

```
protected java.util.List[] listeners
```

Array of `java.util.List`s of `javax.faces.event.ActionListener`s registered for particular phases. The array, as well as the individual elements, are lazily instantiated as necessary.

TYPE

```
public static final java.lang.String TYPE
```

The component type of this `UIComponent` subclass.

Constructors

UICommand()

```
public UICommand()
```

Methods

addActionListener(ActionListener)

```
public void addActionListener(javax.faces.event.ActionListener listener)
```

Add a new `javax.faces.event.ActionListener` to the set of listeners interested in being notified when `javax.faces.event.ActionEvent`s occur.

Parameters:

`listener` - The `javax.faces.event.ActionListener` to be added

Throws:

`NullPointerException` - if `listener` is null

broadcast(ActionEvent, List)

```
protected void broadcast(javax.faces.event.ActionEvent event,
                      java.util.List list)
```

Broadcast the specified `javax.faces.event.ActionEvent` to the `javax.faces.event.ActionListener`s on the specified list (if any)

Parameters:

`event` - The `javax.faces.event.ActionEvent` to be broadcast

`list` - The list of `javax.faces.event.ActionListener`s, or null for no interested listeners

broadcast(FacesEvent, PhaseId)

```
public boolean broadcast(javax.faces.event.FacesEvent event,
                        javax.faces.event.PhaseId phaseId)
                        throws AbortProcessingException
```

Broadcast the specified `javax.faces.event.FacesEvent` to all registered event listeners who have expressed an interest in events of this type, for the specified `javax.faces.event.PhaseId`. The order in which registered listeners are notified is implementation dependent.

After all interested listeners have been notified, return `false` if this event does not have any listeners interested in this event in future phases of the request processing lifecycle. Otherwise, return `true`.

Overrides: `broadcast` in class `UIComponentBase`

Parameters:

`event` - The `javax.faces.event.FacesEvent` to be broadcast

`phaseId` - The `javax.faces.event.PhaseId` of the current phase of the request processing lifecycle

Throws:

`javax.faces.event.AbortProcessingException` - Signal the JavaServer Faces implementation that no further processing on the current event should be performed

`java.lang.IllegalArgumentException` - if the implementation class of this `javax.faces.event.FacesEvent` is not supported by this component
`java.lang.IllegalStateException` - if `PhaseId.ANY_PHASE` is passed for the phase identifier
`NullPointerException` - if event or phaseId is null

decode(FacesContext)

```
public void decode(javax.faces.context.FacesContext context)
                  throws IOException
```

Enqueue a `javax.faces.event.FormEvent` event to the application identifying the form submission that has occurred, along with the command name of the `UICommand` that caused the form to be submitted, if any.

Overrides: `decode` in class `UIComponentBase`

Parameters:

`context` - `javax.faces.context.FacesContext` for the request we are processing

Throws:

`java.io.IOException` - if an input/output error occurs while reading

`NullPointerException` - if `context` is null

encodeEnd(FacesContext)

```
public void encodeEnd(javax.faces.context.FacesContext context)
                      throws IOException
```

Description copied from interface: `javax.faces.component.UIComponent`

Render the ending of the current state of this `UIComponent`, following the rules described for `encodeBegin()` to acquire the appropriate value to be rendered.

Overrides: `encodeEnd` in class `UIComponentBase`

Throws:

`java.io.IOException`

fireActionEvent(FacesContext)

```
public void fireActionEvent(javax.faces.context.FacesContext context)
```

Queue an `javax.faces.event.ActionEvent` for processing during the next event processing cycle.

Parameters:

`context` - The `javax.faces.context.FacesContext` for the current request

getCommandName()

```
public java.lang.String getCommandName()
```

Return the command name for this UICommand_o.

getComponentType()

```
public java.lang.String getComponentType()
```

Description copied from interface: javax.faces.component.UIComponent_o

Return the component type of this UIComponent_o.

Overrides: getComponentType_o in class UIComponentBase_o

removeActionListener(ActionListener)

```
public void removeActionListener(javax.faces.event.ActionListenero listener)
```

Remove an existing javax.faces.event.ActionListener_o (if any) from the set of listeners interested in being notified when javax.faces.event.ActionEvent_os occur.

Parameters:

listener - The javax.faces.event.ActionListener_o to be removed

Throws:

NullPointerException - if listener is null

setCommandName(String)

```
public void setCommandName(java.lang.String commandName)
```

Set the command name for this UICommand_o.

Parameters:

commandName - The new command name

updateModel(FacesContext)

```
public void updateModel(javax.faces.context.FacesContexto context)
```

Override the default behavior and perform no model update.

Overrides: updateModel_o in class UIComponentBase_o

Parameters:

context - FacesContext for the request we are processing

Throws:

java.lang.IllegalArgumentException - if the modelReference property has invalid syntax for an expression

NullPointerException - if context is null

javax.faces.component UIComponent

Declaration

```
public interface UIComponent extends java.io.Serializable
```

All Superinterfaces: `java.io.Serializable`

All Known Implementing Classes: `UIComponentBase0`

Description

UIComponent is the base interface for all user interface components in JavaServer Faces. The set of `UIComponent0` instances associated with a particular request and response are organized into a tree under a root `UIComponent0` that represents the entire content of the request or response.

For the convenience of component developers, `UIComponentBase0` provides the default behavior that is specified for a `UIComponent0`, and is the base class for all of the standard `UIComponent0` implementations. Component writers are encouraged to subclass `UIComponentBase0`, instead of directly implementing this interface, to reduce the impact of any future changes to the method signatures of this interface.

Member Summary

Fields

static <code>CLIENT_ID_ATTR₀</code>	
<code>java.lang.String</code>	The attribute name under which the client identifier for this component is stored, if one has been generated.
static <code>FACET_PARENT₀</code>	
<code>java.lang.String</code>	For <code>UIComponent₀</code> s that are facets, the attribute name under which the owning <code>UIComponent₀</code> is stored.
static <code>FACET_PARENT_ATTR₀</code>	
<code>java.lang.String</code>	For <code>UIComponent₀</code> s that are facets, the attribute name under which the owning <code>UIComponent₀</code> is stored.
static char <code>SEPARATOR_CHAR₀</code>	The separator character used in component identifiers to demarcate navigation to a child naming container.

Methods

Member Summary

void	addChild(int index, UIComponent component)	Insert the specified UIComponent _o at the specified position in the child list for this component.
void	addChild(UIComponent component)	Append the specified UIComponent _o to the end of the child list for this component.
void	addFacet(java.lang.String facetName, UIComponent facet)	Add the specified UIComponent _o as a facet associated with the name specified by the facetName argument, replacing any previous facet with that name.
void	addValidator(javax.faces.validator.Validator validator)	Add a javax.faces.validator.Validator _o instance to the set associated with this UIComponent _o .
boolean	broadcast(javax.faces.event.FacesEvent event, javax.faces.event.PhaseId phaseId)	Broadcast the specified javax.faces.event.FacesEvent _o to all registered event listeners who have expressed an interest in events of this type, for the specified javax.faces.event.PhaseId _o (or for any phase, if the listener returns PhaseId.ANY_PHASE from its getPhaseId() method).
void	clearChildren()	Remove all child UIComponent _o s from the child list.
void	clearFacets()	Remove all facet UIComponent _o s from this component.
void	clearValidators()	Clear any javax.faces.validator.Validator _o s that have been registered for processing by this component.
boolean	containsChild(UIComponent component)	Return true if the specified UIComponent _o is a direct child of this UIComponent _o ; otherwise, return false.
java.lang.Object	currentValue(javax.faces.context.FacesContext context)	Evaluate and return the current value of this component, according to the following algorithm.
void	decode(javax.faces.context.FacesContext context)	Decode the current state of this UIComponent _o from the request contained in the specified javax.faces.context.FacesContext _o , and attempt to convert this state information into an object of the required type for this component (optionally using the registered javax.faces.convert.Converter _o for this component, if there is one).
void	encodeBegin(javax.faces.context.FacesContext context)	Render the beginning of the current state of this UIComponent _o to the response contained in the specified javax.faces.context.FacesContext _o .
void	encodeChildren(javax.faces.context.FacesContext context)	Render the child UIComponent _o s of this UIComponent _o , following the rules described for encodeBegin() to acquire the appropriate value to be rendered.
void	encodeEnd(javax.faces.context.FacesContext context)	Render the ending of the current state of this UIComponent _o , following the rules described for encodeBegin() to acquire the appropriate value to be rendered.

Member Summary

UIComponent	findComponent(java.lang.String expr)	Find the UIComponent _o named by the specified expression, if any is found.
java.lang.Object	getAttribute(java.lang.String name)	Return the value of the attribute with the specified name (if any); otherwise, return null.
java.util.Iterator	getAttributeNames()	Return an Iterator over the names of all currently defined attributes of this UIComponent _o that have a non-null value.
UIComponent	getChild(int index)	Return the UIComponent _o at the specified position in the child list for this component.
int	getChildCount()	Return the number of UIComponent _o s on the child list for this component.
java.util.Iterator	getChildren()	Return an Iterator over the child UIComponent _o s of this UIComponent _o , in the order of their position in the child list.
java.lang.String	getClientId(javax.faces.context.FacesContext context)	Return a client-side identifier for this component.
java.lang.String	getComponentId()	Return the identifier of this UIComponent _o .
java.lang.String	getComponentType()	Return the component type of this UIComponent _o .
java.lang.String	getConverter()	Return the converter id of the javax.faces.convert.Converter _o that is registered for this component.
UIComponent	getFacet(java.lang.String name)	Return the facet UIComponent _o associated with the specified name, if any.
java.util.Iterator	getFacetNames()	Return an Iterator over the names of the facet UIComponent _o s of this UIComponent _o .
java.util.Iterator	getFacetsAndChildren()	Return an Iterator over the facet followed by child UIComponent _o s of this UIComponent _o .
java.lang.String	getModelReference()	Return the model reference expression of this UIComponent _o , if any.
UIComponent	getParent()	Return the parent UIComponent _o of this UIComponent, if any.
java.lang.String	getRendererType()	Return the javax.faces.render.Renderer _o type for this UIComponent _o (if any).
boolean	getRendersChildren()	Return a flag indicating whether this component is responsible for rendering its child components.

Member Summary

<code>boolean getRendersSelf()</code>	Return a flag indicating whether this component has concrete implementations of the <code>decode()</code> and <code>encodeXXX()</code> methods, and is therefore suitable for use in the <i>direct implementation</i> programming model for rendering.
<code>java.util.Iterator getValidators()</code>	Return an Iterator over the <code>javax.faces.validator.Validator</code> s associated with this <code>UIComponent</code> .
<code>java.lang.Object getValue()</code>	Return the local value of this <code>UIComponent</code> , if any.
<code>boolean isRendered()</code>	Return <code>true</code> if the value of the <code>rendered</code> attribute is a Boolean representing <code>true</code> , or is not present; otherwise return <code>false</code> .
<code>boolean isValid()</code>	Return the current validity state of this component.
<code>void processDecodes(javax.faces.context.FacesContext context)</code>	Perform the component tree processing required by the <i>Apply Request Values</i> phase of the request processing lifecycle for all facets of this component, all children of this component, and this component itself, as follows.
<code>void processUpdates(javax.faces.context.FacesContext context)</code>	Perform the component tree processing required by the <i>Update Model Values</i> phase of the request processing lifecycle for all facets of this component, all children of this component, and this component itself, as follows.
<code>void processValidators(javax.faces.context.FacesContext context)</code>	Perform the component tree processing required by the <i>Process Validations</i> phase of the request processing lifecycle for all facets of this component, all children of this component, and this component itself, as follows.
<code>void removeChild(int index)</code>	Remove the child <code>UIComponent</code> at the specified position in the child list for this component.
<code>void removeChild(UIComponent component)</code>	Remove the child <code>UIComponent</code> from the child list for this component.
<code>void removeFacet(java.lang.String name)</code>	Remove the facet <code>UIComponent</code> associated with the specified name, if there is one.
<code>void removeValidator(javax.faces.validator.Validator validator)</code>	Remove a <code>javax.faces.validator.Validator</code> instance from the set associated with this <code>UIComponent</code> , if it was previously associated.
<code>void setAttribute(java.lang.String name, java.lang.Object value)</code>	Set the new value of the attribute with the specified name, replacing any existing value for that name.
<code>void setComponentId(java.lang.String componentId)</code>	Set the identifier of this <code>UIComponent</code> .
<code>void setConverter(java.lang.String converter)</code>	Set the converter id of the <code>javax.faces.convert.Converter</code> that is registered for this component, or null to indicate that there is no registered Converter.

Member Summary

```
void setModelReference(java.lang.String modelReference)
    Set the model reference expression of this UIComponent0.
void setRendered(boolean rendered)
    Set the rendered attribute of this UIComponent0.
void setRendererType(java.lang.String rendererType)
    Set the javax.faces.render.Renderer0 type for this UIComponent0, or
    null for components that render themselves.
void setValid(boolean valid)
    Set the current validity state of this component.
void setValue(java.lang.Object value)
    Set the local value of this UIComponent0.
void updateModel(javax.faces.context.FacesContext context)
    Perform the following algorithm to update the model data associated with this
    UIComponent0, if any, as appropriate.
void validate(javax.faces.context.FacesContext context)
    Perform any correctness checks that this component wishes to perform on itself.
```

Fields

CLIENT_ID_ATTR

```
public static final java.lang.String CLIENT_ID_ATTR
```

The attribute name under which the client identifier for this component is stored, if one has been generated.

FACET_PARENT

```
public static final java.lang.String FACET_PARENT
```

Deprecated. Use FACET_PARENT_ATTR instead

For UIComponent₀s that are facets, the attribute name under which the owning UIComponent₀ is stored.

FACET_PARENT_ATTR

```
public static final java.lang.String FACET_PARENT_ATTR
```

For UIComponent₀s that are facets, the attribute name under which the owning UIComponent₀ is stored.

SEPARATOR_CHAR

```
public static final char SEPARATOR_CHAR
```

The separator character used in component identifiers to demarcate navigation to a child naming container.

Methods

addChild(int, UIComponent)

```
public void addChild(int index, javax.faces.component.UIComponent0 component)
```

Insert the specified `UIComponent0` at the specified position in the child list for this component.

If the child to be added has a non-null and valid component identifier, the identifier is added to the namespace of the closest ancestor that is a naming container.

Parameters:

index - Zero-relative index at which to add this `UIComponent0`

component - The `UIComponent0` to be added

Throws:

`java.lang.IllegalArgumentException` - if the component identifier of the new component is non-null, and is not unique in the namespace of the closest ancestor that is a naming container.

`java.lang.IndexOutOfBoundsException` - if the index is out of range ((index < 0) || (index > size()))

`NullPointerException` - if component is null

addChild(UIComponent)

```
public void addChild(javax.faces.component.UIComponent0 component)
```

Append the specified `UIComponent0` to the end of the child list for this component.

If the child to be added has a non-null and valid component identifier, the identifier is added to the namespace of the closest ancestor that is a naming container.

Parameters:

component - `UIComponent0` to be added

Throws:

`java.lang.IllegalArgumentException` - if the component identifier of the new component is non-null, and is not unique in the namespace of the closest ancestor that is a naming container.

`NullPointerException` - if component is null

addFacet(String, UIComponent)

```
public void addFacet(java.lang.String facetName,  
                     javax.faces.component.UIComponent0 facet)
```

Add the specified `UIComponent0` as a facet associated with the name specified by the `facetName` argument, replacing any previous facet with that name.

This method causes the `FACESET_PARENT_ATTR0` attribute of the specified facet to be set to this `UIComponent0` instance.

Parameters:

`facetName` - The name of this facet

`facet` - The new facet `UIComponent0`

Throws:

`NullPointerException` - if the either of the `facetName` or `facet` arguments are null.

addValidator(Validator)

```
public void addValidator(javax.faces.validator.Validator0 validator)
```

Add a `javax.faces.validator.Validator0` instance to the set associated with this `UIComponent0`.

Parameters:

`validator` - The `javax.faces.validator.Validator0` to add

Throws:

`NullPointerException` - if `validator` is null

broadcast(FacesEvent, PhaseId)

```
public boolean broadcast(javax.faces.event.FacesEvent0 event,  
                        javax.faces.event.PhaseId0 phaseId)  
                        throws AbortProcessingException
```

Broadcast the specified `javax.faces.event.FacesEvent0` to all registered event listeners who have expressed an interest in events of this type, for the specified `javax.faces.event.PhaseId0` (or for any phase, if the listener returns `PhaseId.ANY_PHASE` from its `getPhaseId()` method). The order in which registered listeners are notified is implementation dependent.

After all interested listeners have been notified, return `false` if this event does not have any listeners interested in this event in future phases of the request processing lifecycle. Otherwise, return `true`.

Parameters:

`event` - The `javax.faces.event.FacesEvent0` to be broadcast

`phaseId` - The `javax.faces.event.PhaseId` of the current phase of the request processing lifecycle

Throws:

`javax.faces.event.AbortProcessingException` - Signal the JavaServer Faces implementation that no further processing on the current event should be performed

`java.lang.IllegalArgumentException` - if the implementation class of this `javax.faces.event.FacesEvent` is not supported by this component

`java.lang.IllegalStateException` - if `PhaseId.ANY_PHASE` is passed for the phase identifier

`NullPointerException` - if `event` or `phaseId` is null

clearChildren()

```
public void clearChildren()
```

Remove all child `UIComponent`s from the child list.

clearFacets()

```
public void clearFacets()
```

Remove all facet `UIComponent`s from this component.

clearValidators()

```
public void clearValidators()
```

Clear any `javax.faces.validator.Validator`s that have been registered for processing by this component.

containsChild(`UIComponent`)

```
public boolean containsChild(javax.faces.component.UIComponent component)
```

Return true if the specified `UIComponent` is a direct child of this `UIComponent`; otherwise, return false.

Parameters:

`component` - `UIComponent` to be checked

Throws:

`NullPointerException` - if `component` is null

currentValue(`FacesContext`)

```
public java.lang.Object currentValue(javax.faces.context.FacesContext context)
```

Evaluate and return the current value of this component, according to the following algorithm.

- If the `value` property has been set (containing the local value for this component), return that; else
- If the `modelReference` property has been set, retrieve and return the corresponding model value, if possible; else
- Return `null`.

Parameters:

`context - javax.faces.context.FacesContext0` within which to evaluate the model reference expression, if necessary

Throws:

`NullPointerException` - if `context` is `null`

decode(FacesContext)

```
public void decode(javax.faces.context.FacesContext0 context)
                   throws IOException
```

Decode the current state of this `UIComponent0` from the request contained in the specified `javax.faces.context.FacesContext0`, and attempt to convert this state information into an object of the required type for this component (optionally using the registered `javax.faces.convert.Converter0` for this component, if there is one).

If conversion is successful:

- Save the new local value of this component by calling `setValue()` and passing the new value.
- Set the `valid` property of this component to `true`.

If conversion is not successful:

- Save state information in such a way that encoding can reproduce the previous input (even though it was syntactically or semantically incorrect)
- Add an appropriate conversion failure error message by calling `context.addMessage()`.
- Set the `valid` property of this component to `false`.

During decoding, events may be enqueued for later processing (by event listeners who have registered an interest), by calling `addFacesEvent()` on the associated `javax.faces.context.FacesContext0`.

Parameters:

`context - javax.faces.context.FacesContext0` for the request we are processing

Throws:

`java.io.IOException` - if an input/output error occurs during decoding

NullPointerException - if context is null

encodeBegin(FacesContext)

```
public void encodeBegin(javax.faces.context.FacesContext0 context)
                        throws IOException
```

Render the beginning of the current state of this UIComponent₀ to the response contained in the specified javax.faces.context.FacesContext₀. If the conversion attempted in a previous call to decode() for this component failed, the state information saved during execution of decode() should be utilized to reproduce the incorrect input. If the conversion was successful, or if there was no previous call to decode(), the value to be displayed should be acquired by calling currentValue(), and rendering the value as appropriate.

Parameters:

context - javax.faces.context.FacesContext₀ for the response we are creating

Throws:

java.io.IOException - if an input/output error occurs while rendering

NullPointerException - if context is null

encodeChildren(FacesContext)

```
public void encodeChildren(javax.faces.context.FacesContext0 context)
                           throws IOException
```

Render the child UIComponent₀s of this UIComponent₀, following the rules described for encodeBegin() to acquire the appropriate value to be rendered. This method will only be called if the rendersChildren property is true.

Parameters:

context - javax.faces.context.FacesContext₀ for the response we are creating

Throws:

java.io.IOException - if an input/output error occurs while rendering

NullPointerException - if context is null

encodeEnd(FacesContext)

```
public void encodeEnd(javax.faces.context.FacesContext0 context)
                       throws IOException
```

Render the ending of the current state of this UIComponent₀, following the rules described for encodeBegin() to acquire the appropriate value to be rendered.

Parameters:

context - javax.faces.context.FacesContext₀ for the response we are creating

Throws:

java.io.IOException - if an input/output error occurs while rendering
NullPointerException - if context is null

findComponent(String)

```
public javax.faces.component.UIComponent0 findComponent(java.lang.String expr)
```

Find the UIComponent₀ named by the specified expression, if any is found. This is done by locating the closest parent UIComponent₀ that is a NamingContainer₀, and calling its findComponentInNamespace() method.

The specified expr may contain either a component identifier, or a set of component identifiers separated by SEPARATOR_CHAR characters.

Parameters:

expr - Expression identifying the UIComponent₀ to be returned

Returns: the found UIComponent₀, or null if the component was not found.

Throws:

NullPointerException - if expr is null

getAttribute(String)

```
public java.lang.Object getAttribute(java.lang.String name)
```

Return the value of the attribute with the specified name (if any); otherwise, return null.

Parameters:

name - Name of the requested attribute

Throws:

NullPointerException - if name is null

getAttributeNames()

```
public java.util.Iterator getAttributeNames()
```

Return an Iterator over the names of all currently defined attributes of this UIComponent₀ that have a non-null value.

getChild(int)

```
public javax.faces.component.UIComponent0 getChild(int index)
```

Return the UIComponent₀ at the specified position in the child list for this component.

Parameters:

index - Position of the desired component

Throws:

`java.lang.IndexOutOfBoundsException` - if index is out of range ((index < 0) || (index >= size()))

getCount()

`public int getCount()`

Return the number of `UIComponent`s on the child list for this component.

getChildren()

`public java.util.Iterator getChildren()`

Return an `Iterator` over the child `UIComponent`s of this `UIComponent`, in the order of their position in the child list. If this component has no children, an empty `Iterator` is returned.

getClientId(FacesContext)

`public java.lang.String getClientId(javax.faces.context.FacesContext context)`

Return a client-side identifier for this component.

If a client-side identifier has previously been generated for this component, and saved in the attribute named by `UIComponent.CLIENT_ID`, return that identifier value. Otherwise, generate a new client-side identifier, save it in the attribute named by `UIComponent.CLIENT_ID`, and return it.

Parameters:

`component` - `UIComponent` whose identifier is to be returned

Throws:

`NullPointerException` - if `component` is null

getComponentId()

`public java.lang.String getComponentId()`

Return the identifier of this `UIComponent`.

getComponentType()

`public java.lang.String getComponentType()`

Return the component type of this `UIComponent`.

getConverter()

`public java.lang.String getConverter()`

Return the converter id of the `javax.faces.convert.Converter`₀ that is registered for this component.

getFacet(String)

```
public javax.faces.component.UIComponent0 getFacet(java.lang.String name)
```

Return the facet `UIComponent`₀ associated with the specified name, if any. Otherwise, return null.

Parameters:

name - Name of the facet to be retrieved

Throws:

`NullPointerException` - if name is null

getFacetNames()

```
public java.util.Iterator getFacetNames()
```

Return an `Iterator` over the names of the facet `UIComponent`₀s of this `UIComponent`₀. If this component has no facets, an empty `Iterator` is returned.

getFacetsAndChildren()

```
public java.util.Iterator getFacetsAndChildren()
```

Return an `Iterator` over the facet followed by child `UIComponent`₀s of this `UIComponent`₀. Facets are returned in an undefined order, followed by all the children in the order they are stored in the child list. If this component has no facets or children, an empty `Iterator` is returned.

getModelReference()

```
public java.lang.String getModelReference()
```

Return the model reference expression of this `UIComponent`₀, if any.

getParent()

```
public javax.faces.component.UIComponent0 getParent()
```

Return the parent `UIComponent`₀ of this `UIComponent`, if any.

getRendererType()

```
public java.lang.String getRendererType()
```

Return the `javax.faces.render.Renderer`₀ type for this `UIComponent`₀ (if any).

getRendersChildren()

```
public boolean getRendersChildren()
```

Return a flag indicating whether this component is responsible for rendering its child components.

getRendersSelf()

```
public boolean getRendersSelf()
```

Return a flag indicating whether this component has concrete implementations of the `decode()` and `encodeXXX()` methods, and is therefore suitable for use in the *direct implementation* programming model for rendering.

getValidators()

```
public java.util.Iterator getValidators()
```

Return an `Iterator` over the `javax.faces.validator.Validator`s associated with this `UIComponent`.

getValue()

```
public java.lang.Object getValue()
```

Return the local value of this `UIComponent`, if any.

isRendered()

```
public boolean isRendered()
```

Return `true` if the value of the `rendered` attribute is a Boolean representing `true`, or is not present; otherwise return `false`.

isValid()

```
public boolean isValid()
```

Return the current validity state of this component. The validity state of a component is adjusted at the following points during the request processing lifecycle:

- During the *Apply Request Values* phase, set to `true` or `false` by `decode()`, based on the success or failure of decoding a new local value for this component.
- During the *Process Validations* phase, set to `false` by `processValidations()` if any call to a `validate()` method returned `false`.
- During the *Update Model Values* phase, set to `false` by `updateModel()` if any conversion or update error occurred.

processDecodes(FacesContext)

```
public void processDecodes(javax.faces.context.FacesContext0 context)
    throws IOException
```

Perform the component tree processing required by the *Apply Request Values* phase of the request processing lifecycle for all facets of this component, all children of this component, and this component itself, as follows.

- Call the `processDecodes()` method of all facets and children of this `UIComponent0`, in the order determined by a call to `getFacetsAndChildren()`.
- Call the `decode()` method of this component.

Parameters:

`context - javax.faces.context.FacesContext0` for the request we are processing

Throws:

`java.io.IOException` - if an input/output error occurs during decoding

`NullPointerException` - if `context` is null

processUpdates(FacesContext)

```
public void processUpdates(javax.faces.context.FacesContext0 context)
```

Perform the component tree processing required by the *Update Model Values* phase of the request processing lifecycle for all facets of this component, all children of this component, and this component itself, as follows.

- Call the `processUpdates()` method of all facets and children of this `UIComponent0`, in the order determined by a call to `getFacetsAndChildren()`.
- Call the `updateModel()` method of this component.
- If the `valid` property of this `UIComponent0` is now `false`, call `FacesContext.renderResponse()` to transfer control at the end of the current phase.

Parameters:

`context - javax.faces.context.FacesContext0` for the request we are processing

Throws:

`NullPointerException` - if `context` is null

processValidators(FacesContext)

```
public void processValidators(javax.faces.context.FacesContext0 context)
```

Perform the component tree processing required by the *Process Validations* phase of the request processing lifecycle for all facets of this component, all children of this component, and this component itself, as follows.

- Call the `processValidators()` method of all facets and children of this `UIComponent0`, in the order determined by a call to `getFacetsAndChildren()`.
- If the `valid` property of this component is currently `true`:
 - Call the `validate()` method of each `javax.faces.validator.Validator0` registered for this `UIComponent0`.
 - Call the `validate()` method of this component.
 - Set the `valid` property of this component to the result returned from the `validate()` method.
- If the `valid` property of this `UIComponent0` is now `false`, call `FacesContext.renderResponse()` to transfer control at the end of the current phase.

Parameters:

`context - javax.faces.context.FacesContext0` for the request we are processing

Throws:

`NullPointerException` - if `context` is null

removeChild(int)

```
public void removeChild(int index)
```

Remove the child `UIComponent` at the specified position in the child list for this component.

Parameters:

`index` - Position of the component to be removed

Throws:

`java.lang.IndexOutOfBoundsException` - if the index is out of range (`((index < 0) || (index >= size()))`)

removeChild(UIComponent)

```
public void removeChild(javax.faces.component.UIComponent0 component)
```

Remove the child `UIComponent0` from the child list for this component.

Parameters:

`component` - Child component to be removed

Throws:

`java.lang.IllegalArgumentException` - if `component` is not a child of this component

`NullPointerException` - if `component` is null

removeFacet(String)

```
public void removeFacet(java.lang.String name)
```

Remove the facet `UIComponent`, associated with the specified name, if there is one.

This method causes the `FACET_PARENT_ATTR` attribute of the specified facet to be cleared.

Parameters:

name - Name of the facet to be removed

Throws:

`NullPointerException` - if name is null

removeValidator(Validator)

```
public void removeValidator(javax.faces.validator.Validator, Validator)
```

Remove a `javax.faces.validator.Validator` instance from the set associated with this `UIComponent`, if it was previously associated. Otherwise, do nothing.

Parameters:

validator - The `javax.faces.validator.Validator` to remove

setAttribute(String, Object)

```
public void setAttribute(java.lang.String name, java.lang.Object value)
```

Set the new value of the attribute with the specified name, replacing any existing value for that name.

Parameters:

name - Name of the requested attribute

value - New value (or null to remove any attribute value for the specified name)

Throws:

`java.lang.IllegalArgumentException` - if name represents a read-only property of this component

`NullPointerException` - if name is null

setComponentId(String)

```
public void setComponentId(java.lang.String componentId)
```

Set the identifier of this `UIComponent`.

Parameters:

componentId - The new identifier

Throws:

`java.lang.IllegalArgumentException` - if componentId is zero length or contains invalid characters

`NullPointerException` - if componentId is null

`java.lang.IllegalArgumentException` - if this `UIComponent` instance is already in the tree and and is not unique within the namespace of the closest ancestor that is a naming container.

setConverter(String)

```
public void setConverter(java.lang.String converter)
```

Set the converter id of the `javax.faces.convert.Converter`, that is registered for this component, or null to indicate that there is no registered Converter.

Parameters:

converter - New converter identifier (or null)

setModelReference(String)

```
public void setModelReference(java.lang.String modelReference)
```

Set the model reference expression of this `UIComponent`.

Parameters:

modelReference - The new model reference expression, or null to disconnect this component from any model data

setRendered(boolean)

```
public void setRendered(boolean rendered)
```

Set the rendered attribute of this `UIComponent`.

Parameters:

rendered - If true render this component; otherwise, do not render this component

setRendererType(String)

```
public void setRendererType(java.lang.String rendererType)
```

Set the `javax.faces.render.Renderer` type for this `UIComponent`, or null for components that render themselves.

Parameters:

rendererType - Logical identifier of the type of `javax.faces.render.Renderer` to use, or null for components that render themselves

setValid(boolean)

```
public void setValid(boolean valid)
```

Set the current validity state of this component.

Parameters:

valid - The new validity state

setValue(Object)

```
public void setValue(java.lang.Object value)
```

Set the local value of this `UIComponento`.

Parameters:

value - The new local value, or null for no value

updateModel(FacesContext)

```
public void updateModel(javax.faces.context.FacesContexto context)
```

Perform the following algorithm to update the model data associated with this `UIComponento`, if any, as appropriate.

- If the `valid` property of this component is `false`, take no further action.
- If the `modelReference` property of this component is `null`, take no further action.
- Call the `setModelValue()` method on the specified `javax.faces.context.FacesContexto` instance, passing this component's `modelReference` property and its local value.
- If the `setModelValue()` method returns successfully:
 - Clear the local value of this `UIComponento`.
 - Set the `valid` property of this `UIComponento` to `true`.
- If the `setModelValue()` method call fails:
 - Enqueue error messages by calling `addMessage()` on the specified `javax.faces.context.FacesContexto` instance.
 - Set the `valid` property of this `UIComponento` to `false`.

Parameters:

context - FacesContext for the request we are processing

Throws:

`java.lang.IllegalArgumentException` - if the `modelReference` property has invalid syntax for an expression

NullPointerException - if context is null

validate(FacesContext)

```
public void validate(javax.faces.context.FacesContext0 context)
```

Perform any correctness checks that this component wishes to perform on itself. This method will be called during the *Process Validations* phase of the request processing lifecycle. If errors are encountered, appropriate Message instances should be added to the javax.faces.context.FacesContext₀ for the current request, and the valid property of this UIComponent₀ should be set to false.

Parameters:

context - FacesContext for the request we are processing

Throws:

NullPointerException - if context is null

javax.faces.component UIComponentBase

Declaration

```
public abstract class UIComponentBase implements UIComponent,  
java.lang.Object  
|  
+-- javax.faces.component.UIComponentBase
```

All Implemented Interfaces: java.io.Serializable, UIComponent,

Direct Known Subclasses: UICommand, UIInput, UINamingContainer, UIOutput

Description

UIComponentBase is a convenience base class that implements the default concrete behavior of all methods defined by `UIComponent`, except `getComponentType()`. Concrete subclasses of `UIComponentBase` must implement the `getComponentType()` method. Component types are used to select the appropriate `javax.faces.render.Renderer` to be used for decoding and encoding, if the `rendererType` property is non-null.

By default, this class defines `getRendersChildren()` to return `false`. Subclasses that wish to manage the rendering of their children should override this method to return `true` instead.

By default, this class defines `getRendersSelf()` to return `false`. Subclasses that have directly implemented decoding and encoding functionality, without the requirement for an associated `javax.faces.render.Renderer`, should override this method to return `true` instead.

Member Summary

Constructors

```
UIComponentBase()
```

Methods

```
void addChild(int index, UIComponent component)  
void addChild(UIComponent component)  
void addFacet(java.lang.String facetName, UIComponent facet)  
void addValidator(javax.faces.validator.Validator validator)  
boolean broadcast(javax.faces.event.FacesEvent event,  
                  javax.faces.event.PhaseId phaseId)  
void clearChildren()
```

Member Summary

```
    void clearFacets()
    void clearValidators()
    boolean containsChild(UIComponent component)
    java.lang.Object currentValue(javax.faces.context.FacesContext context)
        void decode(javax.faces.context.FacesContext context)
        void encodeBegin(javax.faces.context.FacesContext context)
        void encodeChildren(javax.faces.context.FacesContext context)
        void encodeEnd(javax.faces.context.FacesContext context)
    UIComponent findComponent(java.lang.String expr)
    java.lang.Object getAttribute(java.lang.String name)
    java.util.Iterator getAttributeNames()
        UIComponent getChild(int index)
            int getChildCount()
    java.util.Iterator getChildren()
    java.lang.String getClientId(javax.faces.context.FacesContext context)
    java.lang.String getComponentId()
        abstract getComponentType()
    java.lang.String getConverter()
    UIComponent getFacet(java.lang.String name)
    java.util.Iterator getFacetNames()
    java.util.Iterator getFacetsAndChildren()
    java.lang.String getModelReference()
        UIComponent getParent()
    java.lang.String getRendererType()
        boolean getRendersChildren()
        boolean getRendersSelf()
    java.util.Iterator getValidators()
    java.lang.Object getValue()
        boolean isRendered()
        boolean isValid()
            void processDecodes(javax.faces.context.FacesContext context)
            void processUpdates(javax.faces.context.FacesContext context)
            void processValidators(javax.faces.context.FacesContext context)
            void removeChild(int index)
            void removeChild(UIComponent component)
            void removeFacet(java.lang.String name)
            void removeValidator(javax.faces.validator.Validator validator)
            void setAttribute(java.lang.String name, java.lang.Object value)
            void setComponentId(java.lang.String componentId)
            void setConverter(java.lang.String converter)
            void setModelReference(java.lang.String modelReference)
            void setRendered(boolean rendered)
            void setRendererType(java.lang.String rendererType)
```

Member Summary

```
void setValid(boolean valid)0
void setValue(java.lang.Object value)0
void updateModel(javax.faces.context.FacesContext context)0
void validate(javax.faces.context.FacesContext context)0
```

Inherited Member Summary**Fields inherited from interface `UIComponent0`**

`CLIENT_ID_ATTR0, FACET_PARENT0, FACET_PARENT_ATTR0, SEPARATOR_CHAR0`

Methods inherited from class `Object`

`clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(),
toString(), wait(), wait(), wait()`

Constructors

`UIComponentBase()`

```
public UIComponentBase()
```

Methods

`addChild(int, UIComponent)`

```
public void addChild(int index, javax.faces.component.UIComponent0 component)
```

Description copied from interface: `javax.faces.component.UIComponent0`

Insert the specified `UIComponent0` at the specified position in the child list for this component.

If the child to be added has a non-null and valid component identifier, the identifier is added to the namespace of the closest ancestor that is a naming container.

Specified By: `addChild0` in interface `UIComponent0`

`addChild(UIComponent)`

```
public void addChild(javax.faces.component.UIComponent0 component)
```

Description copied from interface: javax.faces.component.UIComponent₀

Append the specified UIComponent₀ to the end of the child list for this component.

If the child to be added has a non-null and valid component identifier, the identifier is added to the namespace of the closest ancestor that is a naming container.

Specified By: addChild₀ in interface UIComponent₀

addFacet(String, UIComponent)

```
public void addFacet(java.lang.String facetName,  
                     javax.faces.component.UIComponent0 facet)
```

Description copied from interface: javax.faces.component.UIComponent₀

Add the specified UIComponent₀ as a facet associated with the name specified by the facetName argument, replacing any previous facet with that name.

This method causes the UIComponent.FACET_PARENT_ATTR₀ attribute of the specified facet to be set to this UIComponent₀ instance.

Specified By: addFacet₀ in interface UIComponent₀

addValidator(Validator)

```
public void addValidator(javax.faces.validator.Validator0 validator)
```

Description copied from interface: javax.faces.component.UIComponent₀

Add a javax.faces.validator.Validator₀ instance to the set associated with this UIComponent₀.

Specified By: addValidator₀ in interface UIComponent₀

broadcast(FacesEvent, PhaseId)

```
public boolean broadcast(javax.faces.event.FacesEvent0 event,  
                        javax.faces.event.PhaseId0 phaseId)  
throws AbortProcessingException
```

Description copied from interface: javax.faces.component.UIComponent₀

Broadcast the specified javax.faces.event.FacesEvent₀ to all registered event listeners who have expressed an interest in events of this type, for the specified javax.faces.event.PhaseId₀ (or for any phase, if the listener returns PhaseId.ANY_PHASE from its getPhaseId() method). The order in which registered listeners are notified is implementation dependent.

After all interested listeners have been notified, return `false` if this event does not have any listeners interested in this event in future phases of the request processing lifecycle. Otherwise, return `true`.

Specified By: `broadcast0` in interface `UIComponent0`

Throws:

```
javax.faces.event.AbortProcessingException0
```

clearChildren()

```
public void clearChildren()
```

Description copied from interface: `javax.faces.component.UIComponent0`

Remove all child `UIComponent0`s from the child list.

Specified By: `clearChildren0` in interface `UIComponent0`

clearFacets()

```
public void clearFacets()
```

Description copied from interface: `javax.faces.component.UIComponent0`

Remove all facet `UIComponent0`s from this component.

Specified By: `clearFacets0` in interface `UIComponent0`

clearValidators()

```
public void clearValidators()
```

Description copied from interface: `javax.faces.component.UIComponent0`

Clear any `javax.faces.validator.Validator0`s that have been registered for processing by this component.

Specified By: `clearValidators0` in interface `UIComponent0`

containsChild(UIComponent)

```
public boolean containsChild(javax.faces.component.UIComponent0 component)
```

Description copied from interface: `javax.faces.component.UIComponent0`

Return `true` if the specified `UIComponent0` is a direct child of this `UIComponent0`; otherwise, return `false`.

Specified By: `containsChild0` in interface `UIComponent0`

currentValue(FacesContext)

```
public java.lang.Object currentValue(javax.faces.context.FacesContext0 context)
```

Description copied from interface: javax.faces.component.UIComponent₀

Evaluate and return the current value of this component, according to the following algorithm.

- If the `value` property has been set (containing the local value for this component), return that; else
- If the `modelReference` property has been set, retrieve and return the corresponding model value, if possible; else
- Return `null`.

Specified By: `currentValue0` in interface `UIComponent0`

decode(FacesContext)

```
public void decode(javax.faces.context.FacesContext0 context)
                   throws IOException
```

Description copied from interface: javax.faces.component.UIComponent₀

Decode the current state of this `UIComponent0` from the request contained in the specified `javax.faces.context.FacesContext0`, and attempt to convert this state information into an object of the required type for this component (optionally using the registered `javax.faces.convert.Converter0` for this component, if there is one).

If conversion is successful:

- Save the new local value of this component by calling `setValue()` and passing the new value.
- Set the `valid` property of this component to `true`.

If conversion is not successful:

- Save state information in such a way that encoding can reproduce the previous input (even though it was syntactically or semantically incorrect)
- Add an appropriate conversion failure error message by calling `context.addMessage()`.
- Set the `valid` property of this component to `false`.

During decoding, events may be enqueued for later processing (by event listeners who have registered an interest), by calling `addFacesEvent()` on the associated `javax.faces.context.FacesContext0`.

Specified By: `decode0` in interface `UIComponent0`

Throws:

```
java.io.IOException
```

encodeBegin(FacesContext)

```
public void encodeBegin(javax.faces.context.FacesContext context)
    throws IOException
```

Description copied from interface: javax.faces.component.UIComponent

Render the beginning of the current state of this `UIComponent`, to the response contained in the specified `javax.faces.context.FacesContext`. If the conversion attempted in a previous call to `decode()` for this component failed, the state information saved during execution of `decode()` should be utilized to reproduce the incorrect input. If the conversion was successful, or if there was no previous call to `decode()`, the value to be displayed should be acquired by calling `currentValue()`, and rendering the value as appropriate.

Specified By: `encodeBegin` in interface `UIComponent`

Throws:

```
java.io.IOException
```

encodeChildren(FacesContext)

```
public void encodeChildren(javax.faces.context.FacesContext context)
    throws IOException
```

Description copied from interface: javax.faces.component.UIComponent

Render the child `UIComponent`s of this `UIComponent`, following the rules described for `encodeBegin()` to acquire the appropriate value to be rendered. This method will only be called if the `rendersChildren` property is true.

Specified By: `encodeChildren` in interface `UIComponent`

Throws:

```
java.io.IOException
```

encodeEnd(FacesContext)

```
public void encodeEnd(javax.faces.context.FacesContext context)
    throws IOException
```

Description copied from interface: javax.faces.component.UIComponent

Render the ending of the current state of this `UIComponent`, following the rules described for `encodeBegin()` to acquire the appropriate value to be rendered.

Specified By: encodeEnd₀ in interface UIComponent₀

Throws:

java.io.IOException

findComponent(String)

```
public javax.faces.component.UIComponent0 findComponent(java.lang.String expr)
```

Description copied from interface: javax.faces.component.UIComponent₀

Find the UIComponent₀ named by the specified expression, if any is found. This is done by locating the closest parent UIComponent₀ that is a NamingContainer₀, and calling its findComponentInNamespace() method.

The specified expr may contain either a component identifier, or a set of component identifiers separated by SEPARATOR_CHAR characters.

Specified By: findComponent₀ in interface UIComponent₀

getAttribute(String)

```
public java.lang.Object getAttribute(java.lang.String name)
```

Description copied from interface: javax.faces.component.UIComponent₀

Return the value of the attribute with the specified name (if any); otherwise, return null.

Specified By: getAttribute₀ in interface UIComponent₀

getAttributeNames()

```
public java.util.Iterator getAttributeNames()
```

Description copied from interface: javax.faces.component.UIComponent₀

Return an Iterator over the names of all currently defined attributes of this UIComponent₀ that have a non-null value.

Specified By: getAttributeNames₀ in interface UIComponent₀

getChild(int)

```
public javax.faces.component.UIComponent0 getChild(int index)
```

Description copied from interface: javax.faces.component.UIComponent₀

Return the UIComponent₀ at the specified position in the child list for this component.

Specified By: getChild₀ in interface UIComponent₀

getChildCount()

```
public int getChildCount()
```

Description copied from interface: javax.faces.component.UIComponent₀

Return the number of UIComponent₀s on the child list for this component.

Specified By: getChildCount₀ in interface UIComponent₀

getChildren()

```
public java.util.Iterator getChildren()
```

Description copied from interface: javax.faces.component.UIComponent₀

Return an Iterator over the child UIComponent₀s of this UIComponent₀, in the order of their position in the child list. If this component has no children, an empty Iterator is returned.

Specified By: getChildren₀ in interface UIComponent₀

getClientId(FacesContext)

```
public java.lang.String getClientId(javax.faces.context.FacesContext0 context)
```

Description copied from interface: javax.faces.component.UIComponent₀

Return a client-side identifier for this component.

If a client-side identifier has previously been generated for this component, and saved in the attribute named by UIComponent.CLIENT_ID, return that identifier value. Otherwise, generate a new client-side identifier, save it in the attribute named by UIComponent.CLIENT_ID, and return it.

Specified By: getClientId₀ in interface UIComponent₀

getComponentId()

```
public java.lang.String getComponentId()
```

Description copied from interface: javax.faces.component.UIComponent₀

Return the identifier of this UIComponent₀.

Specified By: getComponentId₀ in interface UIComponent₀

getComponentType()

```
public abstract java.lang.String getComponentType()
```

Description copied from interface: javax.faces.component.UIComponent₀

Return the component type of this UIComponent₀.

Specified By: getComponentType₀ in interface UIComponent₀

getConverter()

```
public java.lang.String getConverter()
```

Description copied from interface: javax.faces.component.UIComponent₀

Return the converter id of the javax.faces.convert.Converter₀ that is registered for this component.

Specified By: getConverter₀ in interface UIComponent₀

getFacet(String)

```
public javax.faces.component.UIComponent0 getFacet(java.lang.String name)
```

Description copied from interface: javax.faces.component.UIComponent₀

Return the facet UIComponent₀ associated with the specified name, if any. Otherwise, return null.

Specified By: getFacet₀ in interface UIComponent₀

getFacetNames()

```
public java.util.Iterator getFacetNames()
```

Description copied from interface: javax.faces.component.UIComponent₀

Return an Iterator over the names of the facet UIComponent₀s of this UIComponent₀. If this component has no facets, an empty Iterator is returned.

Specified By: getFacetNames₀ in interface UIComponent₀

getFacetsAndChildren()

```
public java.util.Iterator getFacetsAndChildren()
```

Description copied from interface: javax.faces.component.UIComponent₀

Return an Iterator over the facet followed by child UIComponent₀s of this UIComponent₀. Facets are returned in an undefined order, followed by all the children in the order they are stored in the child list. If this component has no facets or children, an empty Iterator is returned.

Specified By: `getFacetsAndChildren0` in interface `UIComponent0`

getModelReference()

```
public java.lang.String getModelReference()
```

Description copied from interface: `javax.faces.component.UIComponent0`

Return the model reference expression of this `UIComponent0`, if any.

Specified By: `getModelReference0` in interface `UIComponent0`

getParent()

```
public javax.faces.component.UIComponent0 getParent()
```

Description copied from interface: `javax.faces.component.UIComponent0`

Return the parent `UIComponent0` of this `UIComponent`, if any.

Specified By: `getParent0` in interface `UIComponent0`

getRendererType()

```
public java.lang.String getRendererType()
```

Description copied from interface: `javax.faces.component.UIComponent0`

Return the `javax.faces.render.Renderer0` type for this `UIComponent0` (if any).

Specified By: `getRendererType0` in interface `UIComponent0`

getRendersChildren()

```
public boolean getRendersChildren()
```

Description copied from interface: `javax.faces.component.UIComponent0`

Return a flag indicating whether this component is responsible for rendering its child components.

Specified By: `getRendersChildren0` in interface `UIComponent0`

getRendersSelf()

```
public boolean getRendersSelf()
```

Description copied from interface: javax.faces.component.UIComponent₀

Return a flag indicating whether this component has concrete implementations of the decode() and encodeXXX() methods, and is therefore suitable for use in the *direct implementation* programming model for rendering.

Specified By: getRendersSelf₀ in interface UIComponent₀

getValidators()

```
public java.util.Iterator getValidators()
```

Description copied from interface: javax.faces.component.UIComponent₀

Return an Iterator over the javax.faces.validator.Validator₀s associated with this UIComponent₀.

Specified By: getValidators₀ in interface UIComponent₀

getValue()

```
public java.lang.Object getValue()
```

Description copied from interface: javax.faces.component.UIComponent₀

Return the local value of this UIComponent₀, if any.

Specified By: getValue₀ in interface UIComponent₀

isRendered()

```
public boolean isRendered()
```

Description copied from interface: javax.faces.component.UIComponent₀

Return true if the value of the rendered attribute is a Boolean representing true, or is not present; otherwise return false.

Specified By: isRendered₀ in interface UIComponent₀

isValid()

```
public boolean isValid()
```

Description copied from interface: javax.faces.component.UIComponent₀

Return the current validity state of this component. The validity state of a component is adjusted at the following points during the request processing lifecycle:

- During the *Apply Request Values* phase, set to `true` or `false` by `decode()`, based on the success or failure of decoding a new local value for this component.
- During the *Process Validations* phase, set to `false` by `processValidations()` if any call to a `validate()` method returned `false`.
- During the *Update Model Values* phase, set to `false` by `updateModel()` if any conversion or update error occurred.

Specified By: `isValid0` in interface `UIComponent0`

processDecodes(FacesContext)

```
public void processDecodes(javax.faces.context.FacesContext0 context)
                           throws IOException
```

Description copied from interface: `javax.faces.component.UIComponent0`

Perform the component tree processing required by the *Apply Request Values* phase of the request processing lifecycle for all facets of this component, all children of this component, and this component itself, as follows.

- Call the `processDecodes()` method of all facets and children of this `UIComponent0`, in the order determined by a call to `getFacetsAndChildren()`.
- Call the `decode()` method of this component.

Specified By: `processDecodes0` in interface `UIComponent0`

Throws:

`java.io.IOException`

processUpdates(FacesContext)

```
public void processUpdates(javax.faces.context.FacesContext0 context)
```

Description copied from interface: `javax.faces.component.UIComponent0`

Perform the component tree processing required by the *Update Model Values* phase of the request processing lifecycle for all facets of this component, all children of this component, and this component itself, as follows.

- Call the `processUpdates()` method of all facets and children of this `UIComponent0`, in the order determined by a call to `getFacetsAndChildren()`.
- Call the `updateModel()` method of this component.
- If the `valid` property of this `UIComponent0` is now `false`, call `FacesContext.renderResponse()` to transfer control at the end of the current phase.

Specified By: `processUpdates0` in interface `UIComponent0`

processValidators(FacesContext)

```
public void processValidators(javax.faces.context.FacesContext0 context)
```

Description copied from interface: `javax.faces.component.UIComponent0`

Perform the component tree processing required by the *Process Validations* phase of the request processing lifecycle for all facets of this component, all children of this component, and this component itself, as follows.

- Call the `processValidators()` method of all facets and children of this `UIComponent0`, in the order determined by a call to `getFacetsAndChildren()`.
- If the `valid` property of this component is currently `true`:
 - Call the `validate()` method of each `javax.faces.validator.Validator0` registered for this `UIComponent0`.
 - Call the `validate()` method of this component.
 - Set the `valid` property of this component to the result returned from the `validate()` method.
- If the `valid` property of this `UIComponent0` is now `false`, call `FacesContext.renderResponse()` to transfer control at the end of the current phase.

Specified By: `processValidators0` in interface `UIComponent0`

removeChild(int)

```
public void removeChild(int index)
```

Description copied from interface: `javax.faces.component.UIComponent0`

Remove the child `UIComponent` at the specified position in the child list for this component.

Specified By: `removeChild0` in interface `UIComponent0`

removeChild(UIComponent)

```
public void removeChild(javax.faces.component.UIComponent0 component)
```

Description copied from interface: `javax.faces.component.UIComponent0`

Remove the child `UIComponent0` from the child list for this component.

Specified By: `removeChild0` in interface `UIComponent0`

removeFacet(String)

```
public void removeFacet(java.lang.String name)
```

Description copied from interface: `javax.faces.component.UIComponent`

Remove the facet `UIComponent` associated with the specified name, if there is one.

This method causes the `UIComponent.FACET_PARENT_ATTR` attribute of the specified facet to be cleared.

Specified By: `removeFacet` in interface `UIComponent`

removeValidator(Validator)

```
public void removeValidator(javax.faces.validator.Validator validator)
```

Description copied from interface: `javax.faces.component.UIComponent`

Remove a `javax.faces.validator.Validator` instance from the set associated with this `UIComponent`, if it was previously associated. Otherwise, do nothing.

Specified By: `removeValidator` in interface `UIComponent`

setAttribute(String, Object)

```
public void setAttribute(java.lang.String name, java.lang.Object value)
```

Description copied from interface: `javax.faces.component.UIComponent`

Set the new value of the attribute with the specified name, replacing any existing value for that name.

Specified By: `setAttribute` in interface `UIComponent`

setComponentId(String)

```
public void setComponentId(java.lang.String componentId)
```

Description copied from interface: `javax.faces.component.UIComponent`

Set the identifier of this `UIComponent`.

Specified By: `setComponentId` in interface `UIComponent`

setConverter(String)

```
public void setConverter(java.lang.String converter)
```

Description copied from interface: javax.faces.component.UIComponent₀

Set the converter id of the javax.faces.convert.Converter₀ that is registered for this component, or null to indicate that there is no registered Converter.

Specified By: setConverter₀ in interface UIComponent₀

setModelReference(String)

```
public void setModelReference(java.lang.String modelReference)
```

Description copied from interface: javax.faces.component.UIComponent₀

Set the model reference expression of this UIComponent₀.

Specified By: setModelReference₀ in interface UIComponent₀

setRendered(boolean)

```
public void setRendered(boolean rendered)
```

Description copied from interface: javax.faces.component.UIComponent₀

Set the rendered attribute of this UIComponent₀.

Specified By: setRendered₀ in interface UIComponent₀

setRendererType(String)

```
public void setRendererType(java.lang.String rendererType)
```

Description copied from interface: javax.faces.component.UIComponent₀

Set the javax.faces.render.Renderer₀ type for this UIComponent₀, or null for components that render themselves.

Specified By: setRendererType₀ in interface UIComponent₀

setValid(boolean)

```
public void setValid(boolean valid)
```

Description copied from interface: javax.faces.component.UIComponent₀

Set the current validity state of this component.

Specified By: setValid₀ in interface UIComponent₀

setValue(Object)

```
public void setValue(java.lang.Object value)
```

Description copied from interface: javax.faces.component.UIComponent₀

Set the local value of this UIComponent₀.

Specified By: setValue₀ in interface UIComponent₀

updateModel(FacesContext)

```
public void updateModel(javax.faces.context.FacesContext0 context)
```

Description copied from interface: javax.faces.component.UIComponent₀

Perform the following algorithm to update the model data associated with this UIComponent₀, if any, as appropriate.

- If the valid property of this component is false, take no further action.
- If the modelReference property of this component is null, take no further action.
- Call the setModelValue() method on the specified javax.faces.context.FacesContext₀ instance, passing this component's modelReference property and its local value.
- If the setModelValue() method returns successfully:
 - Clear the local value of this UIComponent₀.
 - Set the valid property of this UIComponent₀ to true.
- If the setModelValue() method call fails:
 - Enqueue error messages by calling addMessage() on the specified javax.faces.context.FacesContext₀ instance.
 - Set the valid property of this UIComponent₀ to false.

Specified By: updateModel₀ in interface UIComponent₀

validate(FacesContext)

```
public void validate(javax.faces.context.FacesContext0 context)
```

Description copied from interface: javax.faces.component.UIComponent₀

Perform any correctness checks that this component wishes to perform on itself. This method will be called during the *Process Validations* phase of the request processing lifecycle. If errors are encountered, appropriate Message instances should be added to the

`javax.faces.context.FacesContext`₀ for the current request, and the `valid` property of this `UIComponent`₀ should be set to `false`.

Specified By: `validate`₀ in interface `UIComponent`₀

javax.faces.component UIForm

Declaration

```
public class UIForm extends UIOutput0  
  
java.lang.Object  
|  
+--javax.faces.component.UIComponentBase0  
|  
+--javax.faces.component.UIOutput0  
|  
+--javax.faces.component.UIForm
```

All Implemented Interfaces: java.io.Serializable, UIComponent₀

Description

UIForm is a UIComponent₀ that represents an input form to be presented to the user, and whose child components represent (among other things) the input fields to be included when the form is submitted.

Member Summary

Fields

```
static TYPE0  
java.lang.String      The component type of this UIComponent0 subclass.
```

Constructors

```
UIForm( )0
```

Methods

```
void encodeBegin(javax.faces.context.FacesContext context)0  
void encodeEnd(javax.faces.context.FacesContext context)0  
java.lang.String getComponentType()0  
java.lang.String getFormName()0  
      Return the form name for this UIForm0.  
void setFormName(java.lang.String formName)0  
      Set the form name for this UIForm0.
```

Inherited Member Summary

Fields inherited from interface `UIComponent`

`CLIENT_ID_ATTR`, `FACET_PARENT`, `FACET_PARENT_ATTR`, `SEPARATOR_CHAR`

Methods inherited from class `Object`

`clone()`, `equals(Object)`, `finalize()`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`,
`toString()`, `wait()`, `wait()`, `wait()`

Methods inherited from class `UIComponentBase`

`addChild(int, UIComponent)`, `addChild(int, UIComponent)`, `addFacet(String, UIComponent)`,
`addValidator(Validator)`, `broadcast(FacesEvent, PhaseId)`,
`clearChildren()`, `clearFacets()`, `clearValidators()`, `containsChild(UIComponent)`,
`currentValue(FacesContext)`, `encodeChildren(FacesContext)`, `findComponent(String)`,
`getAttribute(String)`, `getAttributeNames()`, `getChild(int)`, `getChildCount()`,
`getChildren()`, `getClientId(FacesContext)`, `getComponentId()`, `getConverter()`,
`getFacet(String)`, `getFacetNames()`, `getFacetsAndChildren()`, `getModelReference()`,
`getParent()`, `getRendererType()`, `getRendersChildren()`, `getRendersSelf()`,
`getValidators()`, `getValue()`, `isRendered()`, `isValid()`,
`processDecodes(FacesContext)`, `processUpdates(FacesContext)`,
`processValidators(FacesContext)`, `removeChild(UIComponent)`,
`removeChild(UIComponent)`, `removeFacet(String)`, `removeValidator(Validator)`,
`setAttribute(String, Object)`, `setComponentId(String)`, `setConverter(String)`,
`setModelReference(String)`, `setRendered(boolean)`, `setRendererType(String)`,
`setValid(boolean)`, `setValue(Object)`, `validate(FacesContext)`

Methods inherited from class `UIOutput`

`decode(FacesContext)`, `updateModel(FacesContext)`

Fields

TYPE

`public static final java.lang.String TYPE`

The component type of this `UIComponent` subclass.

Constructors

`UIForm()`

`public UIForm()`

Methods

encodeBegin(FacesContext)

```
public void encodeBegin(javax.faces.context.FacesContext context)
    throws IOException
```

Description copied from interface: javax.faces.component.UIComponent₀

Render the beginning of the current state of this UIComponent₀ to the response contained in the specified javax.faces.context.FacesContext₀. If the conversion attempted in a previous call to decode() for this component failed, the state information saved during execution of decode() should be utilized to reproduce the incorrect input. If the conversion was successful, or if there was no previous call to decode(), the value to be displayed should be acquired by calling currentValue(), and rendering the value as appropriate.

Overrides: encodeBegin₀ in class UIComponentBase₀

Throws:

```
java.io.IOException
```

encodeEnd(FacesContext)

```
public void encodeEnd(javax.faces.context.FacesContext context)
    throws IOException
```

Description copied from interface: javax.faces.component.UIComponent₀

Render the ending of the current state of this UIComponent₀, following the rules described for encodeBegin() to acquire the appropriate value to be rendered.

Overrides: encodeEnd₀ in class UIOutput₀

Throws:

```
java.io.IOException
```

getComponentType()

```
public java.lang.String getComponentType()
```

Description copied from interface: javax.faces.component.UIComponent₀

Return the component type of this UIComponent₀.

Overrides: getComponentType₀ in class UIOutput₀

getFormName()

```
public java.lang.String getFormName()
```

Return the form name for this `UIForm`.

setFormName(String)

```
public void setFormName(java.lang.String formName)
```

Set the form name for this `UIForm`.

Parameters:

`formName` - The new form name

javax.faces.component **UIGraphic**

Declaration

```
public class UIGraphic extends UIOutput0
    java.lang.Object
    |
    +-- javax.faces.component.UIComponentBase0
        |
        +-- javax.faces.component.UIOutput0
            |
            +-- javax.faces.component.UIGraphic
```

All Implemented Interfaces: java.io.Serializable, UIComponent₀

Description

UIGraphic is a UIComponent₀ that displays a graphical image to the user. The user cannot manipulate this component; it is for display purposes only.

Member Summary

Fields

```
static TYPE0
java.lang.String      The component type of this UIComponent0 subclass.
```

Constructors

```
UIGraphic( )0
```

Methods

```
void encodeEnd(javax.faces.context.FacesContext context)0
java.lang.String getComponentType()0
java.lang.String getURL()0
    Return the image URL for this UIGraphic0.
void setURL(java.lang.String url)0
    Set the image URL for this UIGraphic0.
```

Inherited Member Summary

Fields inherited from interface `UIComponent`

`CLIENT_ID_ATTR`, `FACET_PARENT`, `FACET_PARENT_ATTR`, `SEPARATOR_CHAR`

Methods inherited from class `Object`

`clone()`, `equals(Object)`, `finalize()`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`,
`toString()`, `wait()`, `wait()`, `wait()`

Methods inherited from class `UIComponentBase`

`addChild(int, UIComponent)`, `addChild(int, UIComponent)`, `addFacet(String, UIComponent)`,
`addValidator(Validator)`, `broadcast(FacesEvent, PhaseId)`,
`clearChildren()`, `clearFacets()`, `clearValidators()`, `containsChild(UIComponent)`,
`currentValue(FacesContext)`, `encodeBegin(FacesContext)`,
`encodeChildren(FacesContext)`, `findComponent(String)`, `getAttribute(String)`,
`getAttributeNames()`, `getChild(int)`, `getChildCount()`, `getChildren()`,
`getClientId(FacesContext)`, `getComponentId()`, `getConverter()`, `getFacet(String)`,
`getFacetNames()`, `getFacetsAndChildren()`, `getModelReference()`, `getParent()`,
`getRendererType()`, `getRendersChildren()`, `getRendersSelf()`, `getValidators()`,
`getValue()`, `isRendered()`, `isValid()`, `processDecodes(FacesContext)`,
`processUpdates(FacesContext)`, `processValidators(FacesContext)`,
`removeChild(UIComponent)`, `removeChild(UIComponent)`, `removeFacet(String)`,
`removeValidator(Validator)`, `setAttribute(String, Object)`, `setComponentId(String)`,
`setConverter(String)`, `setModelReference(String)`, `setRendered(boolean)`,
`setRendererType(String)`, `setValid(boolean)`, `setValue(Object)`,
`validate(FacesContext)`

Methods inherited from class `UIOutput`

`decode(FacesContext)`, `updateModel(FacesContext)`

Fields

TYPE

`public static final java.lang.String TYPE`

The component type of this `UIComponent` subclass.

Constructors

`UIGraphic()`

`public UIGraphic()`

Methods

encodeEnd(FacesContext)

```
public void encodeEnd(javax.faces.context.FacesContext context)
                      throws IOException
```

Description copied from interface: javax.faces.component.UIComponent

Render the ending of the current state of this UIComponent, following the rules described for encodeBegin() to acquire the appropriate value to be rendered.

Overrides: encodeEnd in class UIOutput

Throws:

```
java.io.IOException
```

getComponentType()

```
public java.lang.String getComponentType()
```

Description copied from interface: javax.faces.component.UIComponent

Return the component type of this UIComponent.

Overrides: getComponentType in class UIOutput

getURL()

```
public java.lang.String getURL()
```

Return the image URL for this UIGraphic.

setURL(String)

```
public void setURL(java.lang.String url)
```

Set the image URL for this UIGraphic.

Parameters:

url - The new image URL

javax.faces.component UIInput

Declaration

```
public class UIInput extends UIComponentBase0
    java.lang.Object
        |
        +-- javax.faces.component.UIComponentBase0
            |
            +-- javax.faces.component.UIInput
```

All Implemented Interfaces: java.io.Serializable, UIComponent₀

Direct Known Subclasses: UISelectBase, UISelectBoolean₀

Description

UIInput is a UIComponent₀ that represents a component that both displays output to the user (like UIOutput₀ components do) and includes request parameters on the subsequent request that need to be decoded. There are no restrictions on the data type of the local value, or the object referenced by the model reference expression (if any); however, individual javax.faces.render.Renderer₀s will generally impose restrictions on the type of data they know how to display.

When the validate() method of this UIInput₀ detects that a value change has actually occurred, and that all validations have been successfully passed, it will queue a javax.faces.event.ValueChangedEvent₀. Later on, the broadcast() method will ensure that this event is broadcast to all interested listeners.

Member Summary

Fields

protected listeners ₀ java.util.List[]	Array of java.util.List s of javax.faces.event.ValueChangedListener ₀ s registered for particular phases.
static PREVIOUS_VALUE ₀ java.lang.String	The symbolic name for the attribute name used to store the previous local value of this UIInput ₀ , saved during decode() processing and used during validate() processing to determine whether a javax.faces.event.ValueChangedEvent ₀ should be queued.

Member Summary

static TYPE₀
java.lang.String The component type of this UIComponent₀ subclass.

Constructors

UIInput()₀

Methods

void addValueChangeListener(javax.faces.event.ValueChangeListener) ₀
Add a new javax.faces.event.ValueChangeListener₀ to the set of listeners interested in being notified when javax.faces.event.ValueChangeEvent₀s occur.

boolean broadcast(javax.faces.event.FacesEvent event, javax.faces.event.PhaseId phaseId)₀
Broadcast the specified javax.faces.event.FacesEvent₀ to all registered event listeners who have expressed an interest in events of this type, for the specified javax.faces.event.PhaseId₀.

protected void broadcast(javax.faces.event.ValueChangeEvent event, java.util.List list)₀
Broadcast the specified javax.faces.event.ValueChangeEvent₀ to the javax.faces.event.ValueChangeListener₀s on the specified list (if any)

protected boolean compareValues(java.lang.Object previous, java.lang.Object value)₀
Return true if the new value is different from the previous value.

void decode(javax.faces.context.FacesContext context)₀

void encodeEnd(javax.faces.context.FacesContext context)₀

protected void fireValueChangedEvent(javax.faces.context.FacesContext context, java.lang.Object oldValue, java.lang.Object newValue)₀
Queue an javax.faces.event.ValueChangeEvent₀ for processing during the next event processing cycle.

java.lang.String getComponentType()₀

void removeValueChangeListener(javax.faces.event.ValueChangeListener) ₀
Remove an existing javax.faces.event.ValueChangeListener₀ (if any) from the set of listeners interested in being notified when javax.faces.event.ValueChangeEvent₀s occur.

void validate(javax.faces.context.FacesContext context)₀
Perform validations and, if validation is successful, queue a javax.faces.event.ValueChangeEvent₀ to be processed later.

Inherited Member Summary

Fields inherited from interface `UIComponent`

`CLIENT_ID_ATTR`, `FACET_PARENT`, `FACET_PARENT_ATTR`, `SEPARATOR_CHAR`

Methods inherited from class `Object`

`clone()`, `equals(Object)`, `finalize()`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `toString()`, `wait()`, `wait()`, `wait()`

Methods inherited from class `UIComponentBase`

`addChild(int, UIComponent)`, `addChild(int, UIComponent)`, `addFacet(String, UIComponent)`, `addValidator(Validator)`, `clearChildren()`, `clearFacets()`, `clearValidators()`, `containsChild(UIComponent)`, `currentValue(FacesContext)`, `encodeBegin(FacesContext)`, `encodeChildren(FacesContext)`, `findComponent(String)`, `getAttribute(String)`, `getAttributeNames()`, `getChild(int)`, `getChildCount()`, `getChildren()`, `getClientId(FacesContext)`, `getComponentId()`, `getConverter()`, `getFacet(String)`, `getFacetNames()`, `getFacetsAndChildren()`, `getModelReference()`, `getParent()`, `getRendererType()`, `getRendersChildren()`, `getRendersSelf()`, `getValidators()`, `getValue()`, `isRendered()`, `isValid()`, `processDecodes(FacesContext)`, `processUpdates(FacesContext)`, `processValidators(FacesContext)`, `removeChild(UIComponent)`, `removeChild(UIComponent)`, `removeFacet(String)`, `removeValidator(Validator)`, `setAttribute(String, Object)`, `setComponentId(String)`, `setConverter(String)`, `setModelReference(String)`, `setRendered(boolean)`, `setRendererType(String)`, `setValid(boolean)`, `setValue(Object)`, `updateModel(FacesContext)`

Fields

listeners

`protected java.util.List[] listeners`

Array of `java.util.List`s of `javax.faces.event.ValueChangedListener`s registered for particular phases. The array, as well as the individual elements, are lazily instantiated as necessary.

PREVIOUS_VALUE

`public static final java.lang.String PREVIOUS_VALUE`

The symbolic name for the attribute name used to store the previous local value of this `UIInput`, saved during `decode()` processing and used during `validate()` processing to determine whether a `javax.faces.event.ValueChangeEvent` should be queued.

TYPE

```
public static final java.lang.String TYPE
```

The component type of this `UIComponent` subclass.

Constructors

UIInput()

```
public UIInput()
```

Methods

addValueChangeListener(ValueChangeListener)

```
public void addValueChangeListener(javax.faces.event.ValueChangeListener  
        listener)
```

Add a new `javax.faces.event.ValueChangeListener` to the set of listeners interested in being notified when `javax.faces.event.ValueChangeEvent`s occur.

Parameters:

listener - The `javax.faces.event.ValueChangeListener` to be added

Throws:

`NullPointerException` - if `listener` is null

broadcast(FacesEvent, PhaseId)

```
public boolean broadcast(javax.faces.event.FacesEvent event,  
        javax.faces.event.PhaseId phaseId)  
throws AbortProcessingException
```

Broadcast the specified `javax.faces.event.FacesEvent` to all registered event listeners who have expressed an interest in events of this type, for the specified `javax.faces.event.PhaseId`. The order in which registered listeners are notified is implementation dependent.

After all interested listeners have been notified, return `false` if this event does not have any listeners interested in this event in future phases of the request processing lifecycle. Otherwise, return `true`.

Overrides: `broadcast` in class `UIComponentBase`

Parameters:

event - The `javax.faces.event.FacesEvent` to be broadcast

`phaseId` - The `javax.faces.event.PhaseId` of the current phase of the request processing lifecycle

Throws:

`javax.faces.event.AbortProcessingException` - Signal the JavaServer Faces implementation that no further processing on the current event should be performed

`java.lang.IllegalArgumentException` - if the implementation class of this `javax.faces.event.FacesEvent` is not supported by this component

`java.lang.IllegalStateException` - if `PhaseId.ANY_PHASE` is passed for the phase identifier

`NullPointerException` - if event or `phaseId` is null

broadcast(ValueChangedEvent, List)

```
protected void broadcast(javax.faces.event.ValueChangedEvent event,
                        java.util.List list)
```

Broadcast the specified `javax.faces.event.ValueChangedEvent` to the `javax.faces.event.ValueChangedListener`s on the specified list (if any)

Parameters:

`event` - The `javax.faces.event.ValueChangedEvent` to be broadcast

`list` - The list of `javax.faces.event.ValueChangedListener`s, or null for no interested listeners

compareValues(Object, Object)

```
protected boolean compareValues(java.lang.Object previous, java.lang.Object value)
```

Return true if the new value is different from the previous value.

Parameters:

`previous` - old value of this component

`value` - new value of this component

decode(FacesContext)

```
public void decode(javax.faces.context.FacesContext context)
                   throws IOException
```

Description copied from interface: `javax.faces.component.UIComponent`

Decode the current state of this `UIComponent` from the request contained in the specified `javax.faces.context.FacesContext`, and attempt to convert this state information into

an object of the required type for this component (optionally using the registered `javax.faces.convert.Converter`₀ for this component, if there is one).

If conversion is successful:

- Save the new local value of this component by calling `setValue()` and passing the new value.
- Set the `valid` property of this component to `true`.

If conversion is not successful:

- Save state information in such a way that encoding can reproduce the previous input (even though it was syntactically or semantically incorrect)
- Add an appropriate conversion failure error message by calling `context.addMessage()`.
- Set the `valid` property of this component to `false`.

During decoding, events may be enqueued for later processing (by event listeners who have registered an interest), by calling `addFacesEvent()` on the associated `javax.faces.context.FacesContext`₀.

Overrides: `decode`₀ in class `UIComponentBase`₀

Throws:

`java.io.IOException`

encodeEnd(FacesContext)

```
public void encodeEnd(javax.faces.context.FacesContext0 context)
                      throws IOException
```

Description copied from interface: `javax.faces.component.UIComponent`₀

Render the ending of the current state of this `UIComponent`₀, following the rules described for `encodeBegin()` to acquire the appropriate value to be rendered.

Overrides: `encodeEnd`₀ in class `UIComponentBase`₀

Throws:

`java.io.IOException`

fireValueChangeEvent(FacesContext, Object, Object)

```
protected void fireValueChangeEvent(javax.faces.context.FacesContext0 context,
                                    java.lang.Object oldValue, java.lang.Object newValue)
```

Queue an `javax.faces.event.ValueChangeEvent`₀ for processing during the next event processing cycle.

Parameters:

context - The javax.faces.context.FacesContext₀ for the current request
oldValue - The original value of this component
newValue - The new value of this component

getComponentType()

```
public java.lang.String getComponentType()
```

Description copied from interface: javax.faces.component.UIComponent₀

Return the component type of this UIComponent₀.

Overrides: getComponentType₀ in class UIComponentBase₀

removeValueChangeListener(ValueChangeListener)

```
public void removeValueChangeListener(javax.faces.event.ValueChangeListener0
    listener)
```

Remove an existing javax.faces.event.ValueChangeListener₀ (if any) from the set of listeners interested in being notified when javax.faces.event.ValueChangeEvent₀s occur.

Parameters:

listener - The javax.faces.event.ValueChangeListener₀ to be removed

Throws:

NullPointerException - if listener is null

validate(FacesContext)

```
public void validate(javax.faces.context.FacesContext0 context)
```

Perform validations and, if validation is successful, queue a javax.faces.event.ValueChangeEvent₀ to be processed later.

Overrides: validate₀ in class UIComponentBase₀

Parameters:

context - The javax.faces.context.FacesContext₀ for the current request

Throws:

NullPointerException - if context is null

javax.faces.component UINamingContainer

Declaration

```
public class UINamingContainer extends UIComponentBase0 implements NamingContainer0  
  
java.lang.Object  
|  
+-- javax.faces.component.UIComponentBase0  
|  
+-- javax.faces.component.UINamingContainer
```

All Implemented Interfaces: NamingContainer₀, java.io.Serializable, UIComponent₀

Description

UINamingContainer is a convenience base class for components that wish to implement NamingContainer₀ functionality.

Member Summary

Fields

```
static TYPE0  
java.lang.String      The component type of this UIComponent0 subclass.
```

Constructors

```
UINamingContainer( )0
```

Methods

```
void addComponentToNamespace(UIComponent namedComponent)0  
UIComponent findComponentInNamespace(java.lang.String name)0  
java.lang.String generateClientId()0  
java.lang.String getComponentType()0  
void removeComponentFromNamespace(UIComponent namedComponent)0
```

Inherited Member Summary

Fields inherited from interface UIComponent₀

Inherited Member Summary

CLIENT_ID_ATTR₀, FACET_PARENT₀, FACET_PARENT_ATTR₀, SEPARATOR_CHAR₀

Methods inherited from class Object

clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(),
toString(), wait(), wait(), wait()

Methods inherited from class UIComponentBase₀

addChild(int, UIComponent)₀, addChild(int, UIComponent)₀, addFacet(String,
UIComponent)₀, addValidator(Validator)₀, broadcast(FacesEvent, PhaseId)₀,
clearChildren()₀, clearFacets()₀, clearValidators()₀, containsChild(UIComponent)₀,
currentValue(FacesContext)₀, decode(FacesContext)₀, encodeBegin(FacesContext)₀,
encodeChildren(FacesContext)₀, encodeEnd(FacesContext)₀, findComponent(String)₀,
getAttribute(String)₀, getAttributeNames()₀, getChild(int)₀, getChildCount()₀,
getChildren()₀, getClientId(FacesContext)₀, getComponentId()₀, getConverter()₀,
getFacet(String)₀, getFacetNames()₀, getFacetsAndChildren()₀, getModelReference()₀,
getParent()₀, getRendererType()₀, getRendersChildren()₀, getRendersSelf()₀,
getValidators()₀, getValue()₀, isRendered()₀, isValid()₀,
processDecodes(FacesContext)₀, processUpdates(FacesContext)₀,
processValidators(FacesContext)₀, removeChild(UIComponent)₀,
removeChild(UIComponent)₀, removeFacet(String)₀, removeValidator(Validator)₀,
setAttribute(String, Object)₀, setComponentId(String)₀, setConverter(String)₀,
setModelReference(String)₀, setRendered(boolean)₀, setRendererType(String)₀,
setValid(boolean)₀, setValue(Object)₀, updateModel(FacesContext)₀,
validate(FacesContext)₀

Fields

TYPE

public static final java.lang.String TYPE

The component type of this UIComponent₀ subclass.

Constructors

UINamingContainer()

public UINamingContainer()

Methods

addComponentToNamespace(UIComponent)

```
public void addComponentToNamespace(javax.faces.component.UIComponent0
                                    namedComponent)
```

Description copied from interface: `javax.faces.component.NamingContainer0`

Add the specified `UIComponent0` to the namespace of this naming container.

Specified By: `addComponentToNamespace0` in interface `NamingContainer0`

findComponentInNamespace(String)

```
public javax.faces.component.UIComponent0
        findComponentInNamespace(java.lang.String name)
```

Description copied from interface: `javax.faces.component.NamingContainer0`

Find and return a `UIComponent0` in this namespace, if it is present; otherwise return null.

If the argument name does not contain any `UIComponent.SEPARATOR_CHAR0` characters, it is interpreted to be a name in the namespace of this naming container.

If the argument name does contain `UIComponent.SEPARATOR_CHAR0` characters, each segment between `UIComponent.SEPARATOR_CHAR0` is treated as a component identifier in its own namespace, which are searched for in child naming containers of this naming container, from left to right.

Specified By: `findComponentInNamespace0` in interface `NamingContainer0`

generateClientId()

```
public java.lang.String generateClientId()
```

Description copied from interface: `javax.faces.component.NamingContainer0`

Generate an identifier for a component, suitable for communication to a client. The identifier returned from this method must be unique within this namespace.

Specified By: `generateClientId0` in interface `NamingContainer0`

getComponentType()

```
public java.lang.String getComponentType()
```

Description copied from interface: javax.faces.component.UIComponent₀

Return the component type of this UIComponent₀.

Specified By: getComponentType₀ in interface UIComponent₀

Overrides: getComponentType₀ in class UIComponentBase₀

removeComponentFromNamespace(UIComponent)

```
public void removeComponentFromNamespace( javax.faces.component.UIComponent0
                                         namedComponent )
```

Description copied from interface: javax.faces.component.NamingContainer₀

Remove the specified UIComponent₀ from the namespace of this naming container, if it is present.

Specified By: removeComponentFromNamespace₀ in interface NamingContainer₀

javax.faces.component UIOutput

Declaration

```
public class UIOutput extends UIComponentBase,  
    java.lang.Object  
    |  
    +-- javax.faces.component.UIComponentBase,  
    |  
    +-- javax.faces.component.UIOutput
```

All Implemented Interfaces: `java.io.Serializable, UIComponent`,

Direct Known Subclasses: `UIForm`, `UIGraphic`, `UIPanel`, `UIParameter`,
`UISelectItem`, `UISelectItems`

Description

UIOutput is a `UIComponent` that displays output to the user. The user cannot manipulate this component; it is for display purposes only. There are no restrictions on the data type of the local value, or the object referenced by the model reference expression (if any); however, individual `javax.faces.render.Renderer`s will generally impose restrictions on the type of data they know how to display.

Member Summary

Fields

```
static TYPE  
java.lang.String      The component type of this UIComponent subclass.
```

Constructors

```
UIOutput()
```

Methods

```
void decode(javax.faces.context.FacesContext context)  
    Set the valid property to true, and perform no decoding.
```

```
void encodeEnd(javax.faces.context.FacesContext context)
```

```
java.lang.String getComponentType()
```

```
void updateModel(javax.faces.context.FacesContext context)  
    Override the default behavior and perform no model update.
```

Inherited Member Summary
Fields inherited from interface <code>UIComponent</code>
<code>CLIENT_ID_ATTR</code> , <code>FACET_PARENT</code> , <code>FACET_PARENT_ATTR</code> , <code>SEPARATOR_CHAR</code>
Methods inherited from class <code>Object</code>
<code>clone()</code> , <code>equals(Object)</code> , <code>finalize()</code> , <code>getClass()</code> , <code>hashCode()</code> , <code>notify()</code> , <code>notifyAll()</code> , <code>toString()</code> , <code>wait()</code> , <code>wait()</code> , <code>wait()</code>
Methods inherited from class <code>UIComponentBase</code>
<code>addChild(int, UIComponent)</code> , <code>addChild(int, UIComponent)</code> , <code>addFacet(String, UIComponent)</code> , <code>addValidator(Validator)</code> , <code>broadcast(FacesEvent, PhaseId)</code> , <code>clearChildren()</code> , <code>clearFacets()</code> , <code>clearValidators()</code> , <code>containsChild(UIComponent)</code> , <code>currentValue(FacesContext)</code> , <code>encodeBegin(FacesContext)</code> , <code>encodeChildren(FacesContext)</code> , <code>findComponent(String)</code> , <code>getAttribute(String)</code> , <code>getAttributeNames()</code> , <code>getChild(int)</code> , <code>getChildCount()</code> , <code>getChildren()</code> , <code>getClientId(FacesContext)</code> , <code>getComponentId()</code> , <code>getConverter()</code> , <code>getFacet(String)</code> , <code>getFacetNames()</code> , <code>getFacetsAndChildren()</code> , <code>getModelReference()</code> , <code>getParent()</code> , <code>getRendererType()</code> , <code>getRendersChildren()</code> , <code>getRendersSelf()</code> , <code>getValidators()</code> , <code>getValue()</code> , <code>isRendered()</code> , <code>isValid()</code> , <code>processDecodes(FacesContext)</code> , <code>processUpdates(FacesContext)</code> , <code>processValidators(FacesContext)</code> , <code>removeChild(UIComponent)</code> , <code>removeChild(UIComponent)</code> , <code>removeFacet(String)</code> , <code>removeValidator(Validator)</code> , <code>setAttribute(String, Object)</code> , <code>setComponentId(String)</code> , <code>setConverter(String)</code> , <code>setModelReference(String)</code> , <code>setRendered(boolean)</code> , <code>setRendererType(String)</code> , <code>setValid(boolean)</code> , <code>setValue(Object)</code> , <code>validate(FacesContext)</code>

Fields

TYPE

```
public static final java.lang.String TYPE
```

The component type of this `UIComponent` subclass.

Constructors

`UIOutput()`

```
public UIOutput()
```

Methods

decode(FacesContext)

```
public void decode(javax.faces.context.FacesContext0 context)
                   throws IOException
```

Set the valid property to true, and perform no decoding.

Overrides: decode₀ in class UIComponentBase₀

Parameters:

context - javax.faces.context.FacesContext₀ for the request we are processing

Throws:

java.io.IOException - if an input/output error occurs while reading

NullPointerException - if context is null

encodeEnd(FacesContext)

```
public void encodeEnd(javax.faces.context.FacesContext0 context)
                      throws IOException
```

Description copied from interface: javax.faces.component.UIComponent₀

Render the ending of the current state of this UIComponent₀, following the rules described for encodeBegin() to acquire the appropriate value to be rendered.

Overrides: encodeEnd₀ in class UIComponentBase₀

Throws:

java.io.IOException

getComponentType()

```
public java.lang.String getComponentType()
```

Description copied from interface: javax.faces.component.UIComponent₀

Return the component type of this UIComponent₀.

Overrides: getComponentType₀ in class UIComponentBase₀

updateModel(FacesContext)

```
public void updateModel(javax.faces.context.FacesContext0 context)
```

Override the default behavior and perform no model update.

Overrides: updateModel₀ in class UIComponentBase₀

Parameters:

context - FacesContext for the request we are processing

Throws:

java.lang.IllegalArgumentException - if the modelReference property has invalid syntax for an expression

NullPointerException - if context is null

javax.faces.component UIPanel

Declaration

```
public class UIPanel extends UIOutput,  
  
java.lang.Object  
|  
+--javax.faces.component.UIComponentBase,  
|  
+--javax.faces.component.UIOutput,  
|  
+--javax.faces.component.UIPanel
```

All Implemented Interfaces: java.io.Serializable, UIComponent,

Description

UIPanel is a `UIComponent` that manages the layout of its child components.

Member Summary

Fields

```
static TYPE  
java.lang.String      The component type of this UIComponent subclass.
```

Constructors

```
UIPanel()
```

Methods

```
java.lang.String  getComponentType()  
boolean  getRendersChildren()  
Return true to indicate that this component takes responsibility for rendering its  
children.
```

Inherited Member Summary

Fields inherited from interface `UIComponent`

```
CLIENT_ID_ATTR, FACET_PARENT, FACET_PARENT_ATTR, SEPARATOR_CHAR
```

Inherited Member Summary

Methods inherited from class `Object`

`clone()`, `equals(Object)`, `finalize()`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`,
`toString()`, `wait()`, `wait()`, `wait()`

Methods inherited from class `UIComponentBase`

`addChild(int, UIComponent)`, `addChild(int, UIComponent)`, `addFacet(String, UIComponent)`,
`addValidator(Validator)`, `broadcast(FacesEvent, PhaseId)`,
`clearChildren()`, `clearFacets()`, `clearValidators()`, `containsChild(UIComponent)`,
`currentValue(FacesContext)`, `encodeBegin(FacesContext)`,
`encodeChildren(FacesContext)`, `findComponent(String)`, `getAttribute(String)`,
`getAttributeNames()`, `getChild(int)`, `getChildCount()`, `getChildren()`,
`getClientId(FacesContext)`, `getComponentId()`, `getConverter()`, `getFacet(String)`,
`getFacetNames()`, `getFacetsAndChildren()`, `getModelReference()`, `getParent()`,
`getRendererType()`, `getRendersSelf()`, `getValidators()`, `getValue()`, `isRendered()`,
`isValid()`, `processDecodes(FacesContext)`, `processUpdates(FacesContext)`,
`processValidators(FacesContext)`, `removeChild(UIComponent)`,
`removeChild(UIComponent)`, `removeFacet(String)`, `removeValidator(Validator)`,
`setAttribute(String, Object)`, `setComponentId(String)`, `setConverter(String)`,
`setModelReference(String)`, `setRendered(boolean)`, `setRendererType(String)`,
`setValid(boolean)`, `setValue(Object)`, `validate(FacesContext)`

Methods inherited from class `UIOutput`

`decode(FacesContext)`, `encodeEnd(FacesContext)`, `updateModel(FacesContext)`

Fields

`TYPE`

`public static final java.lang.String TYPE`

The component type of this `UIComponent` subclass.

Constructors

`UIPanel()`

`public UIPanel()`

Methods

getComponentType()

```
public java.lang.String getComponentType()
```

Description copied from interface: `javax.faces.component.UIComponent`

Return the component type of this `UIComponent`.

Overrides: `getComponentType` in class `UIOutput`

getRendersChildren()

```
public boolean getRendersChildren()
```

Return `true` to indicate that this component takes responsibility for rendering its children.

Overrides: `getRendersChildren` in class `UIComponentBase`

javax.faces.component UIParameter

Declaration

```
public class UIParameter extends UIOutput0
    java.lang.Object
        |
        +-- javax.faces.component.UIComponentBase0
            |
            +-- javax.faces.component.UIOutput0
                |
                +-- javax.faces.component.UIParameter
```

All Implemented Interfaces: java.io.Serializable, UIComponent₀

Description

UIParameter is a UIComponent₀ that represents an optionally named configuration parameter for a parent component.

Parent components should retrieve the value of a parameter by calling `currentValue()`. In this way, the parameter value can be set directly on the component (via `setValue()`), or retrieved indirectly via the model reference expression.

In some scenarios, it is necessary to provide a parameter name, in addition to the parameter value that is accessible via the `currentValue()` method. `javax.faces.render.Renderer`s that support parameter names on their nested UIParameter₀ child components should document their use of this property.

Member Summary

Fields

static TYPE ₀	
java.lang.String	The component type of this UIComponent ₀ subclass.

Constructors

UIParameter() ₀

Methods

void encodeBegin(javax.faces.context.FacesContext context) ₀	Override the default behavior and perform no encoding.
void encodeChildren(javax.faces.context.FacesContext context) ₀	Override the default behavior and perform no encoding.

Member Summary

```
void encodeEnd(javax.faces.context.FacesContext context)
    Override the default behavior and perform no encoding.

java.lang.String getComponentType()
java.lang.String getName()
    Return the optional parameter name for this parameter.

boolean getRendersSelf()
    Return true to indicate that no javax.faces.render.Renderer
        needs to be associated with this component.

void setName(java.lang.String name)
    Set the optional parameter name for this parameter.
```

Inherited Member Summary**Fields inherited from interface UIComponent₀**

CLIENT_ID_ATTR₀, FACET_PARENT₀, FACET_PARENT_ATTR₀, SEPARATOR_CHAR₀

Methods inherited from class Object

clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(),
toString(), wait(), wait(), wait()

Methods inherited from class UIComponentBase₀

addChild(int, UIComponent)₀, addChild(int, UIComponent)₀, addFacet(String,
UIComponent)₀, addValidator(Validator)₀, broadcast(FacesEvent, PhaseId)₀,
clearChildren()₀, clearFacets()₀, clearValidators()₀, containsChild(UIComponent)₀,
currentValue(FacesContext)₀, findComponent(String)₀, getAttribute(String)₀,
getAttributeNames()₀, getChild(int)₀, getChildCount()₀, getChildren()₀,
getClientId(FacesContext)₀, getComponentId()₀, getConverter()₀, getFacet(String)₀,
getFacetNames()₀, getFacetsAndChildren()₀, getModelReference()₀, getParent()₀,
getRendererType()₀, getRendersChildren()₀, getValidators()₀, getValue()₀,
isRendered()₀, isValid()₀, processDecodes(FacesContext)₀,
processUpdates(FacesContext)₀, processValidators(FacesContext)₀,
removeChild(UIComponent)₀, removeChild(UIComponent)₀, removeFacet(String)₀,
removeValidator(Validator)₀, setAttribute(String, Object)₀, setComponentId(String)₀,
setConverter(String)₀, setModelReference(String)₀, setRendered(boolean)₀,
setRendererType(String)₀, setValid(boolean)₀, setValue(Object)₀,
validate(FacesContext)₀

Methods inherited from class UIOutput₀

decode(FacesContext)₀, updateModel(FacesContext)₀

Fields

TYPE

```
public static final java.lang.String TYPE
```

The component type of this `UIComponent0` subclass.

Constructors

UIParameter()

```
public UIParameter()
```

Methods

encodeBegin(FacesContext)

```
public void encodeBegin(javax.faces.context.FacesContext0 context)
                        throws IOException
```

Override the default behavior and perform no encoding.

Overrides: `encodeBegin0` in class `UIComponentBase0`

Parameters:

context - `javax.faces.context.FacesContext0` for the response we are creating

Throws:

`java.io.IOException` - if an input/output error occurs while rendering

`NullPointerException` - if context is null

encodeChildren(FacesContext)

```
public void encodeChildren(javax.faces.context.FacesContext0 context)
                           throws IOException
```

Override the default behavior and perform no encoding.

Overrides: `encodeChildren0` in class `UIComponentBase0`

Parameters:

context - `javax.faces.context.FacesContext0` for the response we are creating

Throws:

`java.io.IOException` - if an input/output error occurs while rendering

`NullPointerException` - if context is null

encodeEnd(FacesContext)

```
public void encodeEnd(javax.faces.context.FacesContext0 context)
                      throws IOException
```

Override the default behavior and perform no encoding.

Overrides: encodeEnd₀ in class UIOutput₀

Parameters:

context - javax.faces.context.FacesContext₀ for the response we are creating

Throws:

java.io.IOException - if an input/output error occurs while rendering

NullPointerException - if context is null

getComponentType()

```
public java.lang.String getComponentType()
```

Description copied from interface: javax.faces.component.UIComponent₀

Return the component type of this UIComponent₀.

Overrides: getComponentType₀ in class UIOutput₀

getName()

```
public java.lang.String getName()
```

Return the optional parameter name for this parameter.

getRendersSelf()

```
public boolean getRendersSelf()
```

Return true to indicate that no javax.faces.render.Renderer₀ needs to be associated with this component.

Overrides: getRendersSelf₀ in class UIComponentBase₀

setName(String)

```
public void setName(java.lang.String name)
```

Set the optional parameter name for this parameter.

Parameters:

name - The new parameter name, or null for no name

javax.faces.component UISelectBoolean

Declaration

```
public class UISelectBoolean extends UIInput  
  
java.lang.Object  
|  
+--javax.faces.component.UIComponentBase  
|  
+--javax.faces.component.UIInput  
|  
+--javax.faces.component.UISelectBoolean
```

All Implemented Interfaces: java.io.Serializable, UIComponent

Description

UISelectBoolean is a `UIComponent` that represents a single boolean (`true` or `false`) value. It is most commonly rendered as a checkbox.

Member Summary

Fields

```
static TYPE  
java.lang.String      The component type of this UIComponent subclass.
```

Constructors

```
UISelectBoolean()
```

Methods

```
void decode(javax.faces.context.FacesContext context)  
void encodeEnd(javax.faces.context.FacesContext context)  
java.lang.String getComponentType()  
boolean isSelected()  
                                         Return the local value of the selected state of this component.  
void setSelected(boolean selected)  
                                         Set the local value of the selected state of this component.
```

Inherited Member Summary

Fields inherited from interface `UIComponent`

`CLIENT_ID_ATTR0, FACET_PARENT0, FACET_PARENT_ATTR0, SEPARATOR_CHAR0`

Fields inherited from class `UIInput`

`PREVIOUS_VALUE0, listeners0`

Methods inherited from class `Object`

`clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(),
toString(), wait(), wait(), wait()`

Methods inherited from class `UIComponentBase`

`addChild(int, UIComponent)0, addChild(int, UIComponent)0, addFacet(String,
UIComponent)0, addValidator(Validator)0, clearChildren()0, clearFacets()0,
clearValidators()0, containsChild(UIComponent)0, currentValue(FacesContext)0,
encodeBegin(FacesContext)0, encodeChildren(FacesContext)0, findComponent(String)0,
getAttribute(String)0, getAttributeNames()0, getChild(int)0, getChildCount()0,
getChildren()0, getClientId(FacesContext)0, getComponentId()0, getConverter()0,
getFacet(String)0, getFacetNames()0, getFacetsAndChildren()0, getModelReference()0,
getParent()0, getRendererType()0, getRendersChildren()0, getRendersSelf()0,
getValidators()0, getValue()0, isRendered()0, isValid()0,
processDecodes(FacesContext)0, processUpdates(FacesContext)0,
processValidators(FacesContext)0, removeChild(UIComponent)0,
removeChild(UIComponent)0, removeFacet(String)0, removeValidator(Validator)0,
setAttribute(String, Object)0, setComponentId(String)0, setConverter(String)0,
setModelReference(String)0, setRendered(boolean)0, setRendererType(String)0,
setValid(boolean)0, setValue(Object)0, updateModel(FacesContext)0`

Methods inherited from class `UIInput`

`addValueChangeListener(ValueChangedListener)0, broadcast(ValueChangedEvent, List)0,
broadcast(ValueChangedEvent, List)0, compareValues(Object, Object)0,
fireValueChangeEvent(FacesContext, Object, Object)0,
removeValueChangeListener(ValueChangedListener)0, validate(FacesContext)0`

Fields

TYPE

`public static final java.lang.String TYPE`

The component type of this `UIComponent0` subclass.

Constructors

UISelectBoolean()

```
public UISelectBoolean()
```

Methods

decode(FacesContext)

```
public void decode(javax.faces.context.FacesContext context)
    throws IOException
```

Description copied from interface: `javax.faces.component.UIComponent`

Decode the current state of this `UIComponent` from the request contained in the specified `javax.faces.context.FacesContext`, and attempt to convert this state information into an object of the required type for this component (optionally using the registered `javax.faces.convert.Converter` for this component, if there is one).

If conversion is successful:

- Save the new local value of this component by calling `setValue()` and passing the new value.
- Set the `valid` property of this component to `true`.

If conversion is not successful:

- Save state information in such a way that encoding can reproduce the previous input (even though it was syntactically or semantically incorrect)
- Add an appropriate conversion failure error message by calling `context.addMessage()`.
- Set the `valid` property of this component to `false`.

During decoding, events may be enqueued for later processing (by event listeners who have registered an interest), by calling `addFacesEvent()` on the associated `javax.faces.context.FacesContext`.

Overrides: `decode` in class `UIInput`

Throws:

`java.io.IOException`

encodeEnd(FacesContext)

```
public void encodeEnd(javax.faces.context.FacesContext context)
    throws IOException
```

Description copied from interface: javax.faces.component.UIComponent₀

Render the ending of the current state of this UIComponent₀, following the rules described for encodeBegin() to acquire the appropriate value to be rendered.

Overrides: encodeEnd₀ in class UIInput₀

Throws:

java.io.IOException

getComponentType()

```
public java.lang.String getComponentType()
```

Description copied from interface: javax.faces.component.UIComponent₀

Return the component type of this UIComponent₀.

Overrides: getComponentType₀ in class UIInput₀

isSelected()

```
public boolean isSelected()
```

Return the local value of the selected state of this component.

setSelected(boolean)

```
public void setSelected(boolean selected)
```

Set the local value of the selected state of this component.

Parameters:

selected - The new selected state

javax.faces.component UISelectItem

Declaration

```
public class UISelectItem extends UIOutput0
java.lang.Object
|
+--javax.faces.component.UIComponentBase0
|
+--javax.faces.component.UIOutput0
|
+--javax.faces.component.UISelectItem
```

All Implemented Interfaces: java.io.Serializable, UIComponent₀

Description

UISelectItem is a component that may be nested inside a `UISelectMany0` or `UISelectOne0` component, and causes the addition of a `SelectItem0` instance to the list of available options for the parent component. The contents of the `SelectItem0` can be specified in one of the following ways:

- The `value` attribute's value is an instance of `SelectItem0`.
- The `modelReference` attribute points at a model data item of type `SelectItem0`.
- A new `SelectItem0` instance is synthesized from the values of the `itemDescription`, `itemLabel`, and `itemValue` attributes.

Member Summary

Fields

static TYPE ₀	
java.lang.String	The component type of this <code>UIComponent₀</code> subclass.

Constructors

UISelectItem()	
----------------	--

Methods

void encodeBegin(javax.faces.context.FacesContext context)	
	Override the default behavior and perform no encoding.
void encodeChildren(javax.faces.context.FacesContext context)	
	Override the default behavior and perform no encoding.

Member Summary

```
void encodeEnd(javax.faces.context.FacesContext context)
Override the default behavior and perform no encoding.

java.lang.String getComponentType()
java.lang.String getItemDescription()
Return the item description for this component (if value and modelReference
are null).

java.lang.String getItemLabel()
Return the item label for this component (if value and modelReference are
null).

java.lang.String getItemValue()
Return the item value for this component (if value and modelReference are
null).

boolean getRendersSelf()
Return true to indicate that no javax.faces.render.Renderer needs to be
associated with this component.

void setItemDescription(java.lang.String itemDescription)
Set the item description for this component (if value and modelReference are
null).

void setItemLabel(java.lang.String itemLabel)
Set the item label for this component (if value and modelReference are null).

void setItemValue(java.lang.String itemValue)
Set the item value for this component (if value and modelReference are null).
```

Inherited Member Summary**Fields inherited from interface `UIComponent`**

CLIENT_ID_ATTR, FACET_PARENT, FACET_PARENT_ATTR, SEPARATOR_CHAR

Methods inherited from class `Object`

clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(),
toString(), wait(), wait(), wait()

Methods inherited from class `UIComponentBase`

Inherited Member Summary

```
addChild(int, UIComponent)o, addChild(int, UIComponent)o, addFacet(String,  
UIComponent)o, addValidator(Validator)o, broadcast(FacesEvent, PhaseId)o,  
clearChildren()o, clearFacets()o, clearValidators()o, containsChild(UIComponent)o,  
currentValue(FacesContext)o, findComponent(String)o, getAttribute(String)o,  
getAttributeNames()o, getChild(int)o, getChildCount()o, getChildren()o,  
getClientId(FacesContext)o, getComponentId()o, getConverter()o, getFacet(String)o,  
getFacetNames()o, getFacetsAndChildren()o, getModelReference()o, getParent()o,  
getRendererType()o, getRendersChildren()o, getValidators()o, getValue()o,  
isRendered()o, isValid()o, processDecodes(FacesContext)o,  
processUpdates(FacesContext)o, processValidators(FacesContext)o,  
removeChild(UIComponent)o, removeChild(UIComponent)o, removeFacet(String)o,  
removeValidator(Validator)o, setAttribute(String, Object)o, setComponentId(String)o,  
setConverter(String)o, setModelReference(String)o, setRendered(boolean)o,  
setRendererType(String)o, setValid(boolean)o, setValue(Object)o,  
validate(FacesContext)o
```

Methods inherited from class `UIOutput`

```
decode(FacesContext)o, updateModel(FacesContext)o
```

Fields

TYPE

```
public static final java.lang.String TYPE
```

The component type of this `UIComponent` subclass.

Constructors

`UISelectItem()`

```
public UISelectItem()
```

Methods

`encodeBegin(FacesContext)`

```
public void encodeBegin(javax.faces.context.FacesContext o context)  
throws IOException
```

Override the default behavior and perform no encoding.

Overrides: `encodeBegin` in class `UIComponentBase`

Parameters:

context - javax.faces.context.FacesContext₀ for the request we are processing

Throws:

java.io.IOException - if an input/output error occurs while reading

NullPointerException - if context is null

encodeChildren(FacesContext)

```
public void encodeChildren(javax.faces.context.FacesContext0 context)
                           throws IOException
```

Override the default behavior and perform no encoding.

Overrides: encodeChildren₀ in class UIComponentBase₀

Parameters:

context - javax.faces.context.FacesContext₀ for the request we are processing

Throws:

java.io.IOException - if an input/output error occurs while reading

NullPointerException - if context is null

encodeEnd(FacesContext)

```
public void encodeEnd(javax.faces.context.FacesContext0 context)
                       throws IOException
```

Override the default behavior and perform no encoding.

Overrides: encodeEnd₀ in class UIOutput₀

Parameters:

context - javax.faces.context.FacesContext₀ for the request we are processing

Throws:

java.io.IOException - if an input/output error occurs while reading

NullPointerException - if context is null

getComponentType()

```
public java.lang.String getComponentType()
```

Description copied from interface: javax.faces.component.UIComponent₀

Return the component type of this UIComponent₀.

Overrides: getComponentType₀ in class UIOutput₀

getItemDescription()

```
public java.lang.String getItemDescription()
```

Return the item description for this component (if value and modelReference are null).

getItemLabel()

```
public java.lang.String getItemLabel()
```

Return the item label for this component (if value and modelReference are null).

getItemValue()

```
public java.lang.String getItemValue()
```

Return the item value for this component (if value and modelReference are null).

getRendersSelf()

```
public boolean getRendersSelf()
```

Return true to indicate that no javax.faces.render.Renderer₀ needs to be associated with this component.

Overrides: getRendersSelf₀ in class UIComponentBase₀

setItemDescription(String)

```
public void setItemDescription(java.lang.String itemDescription)
```

Set the item description for this component (if value and modelReference are null).

setItemLabel(String)

```
public void setItemLabel(java.lang.String itemLabel)
```

Set the item label for this component (if value and modelReference are null).

setItemValue(String)

```
public void setItemValue(java.lang.String itemValue)
```

Set the item value for this component (if value and modelReference are null).

javax.faces.component UISelectItems

Declaration

```
public class UISelectItems extends UIOutput,  
  
java.lang.Object  
|  
+--javax.faces.component.UIComponentBase,  
|  
+--javax.faces.component.UIOutput,  
|  
+--javax.faces.component.UISelectItems
```

All Implemented Interfaces: java.io.Serializable, UIComponent,

Description

UISelectItems is a component that may be nested inside a **UISelectMany** or **UISelectOne** component, and causes the addition of one or more **SelectItem** instances to the list of available options in the parent component. The value of this component (set either directly, or acquired indirectly via the `modelReference` property, can be of any of the following types:

- *Single instance of SelectItem* - This instance is added to the set of available options for the parent tag.
- *Array of SelectItem* - This set of instances is added to the set of available options for the parent component, in ascending subscript order.
- *Collection of SelectItem* - This set of instances is added to the set of available options for the parent component, in the order provided by an iterator over them.
- *Map* - The keys of this object (once converted to Strings) are assumed to be labels, and the values of this object (once converted to Strings) are assumed to be values, of **SelectItem** instances that will be constructed dynamically and added to the set of available options for the parent component, in the order provided by an iterator over the keys.

Member Summary

Fields

static TYPE	
java.lang.String	The component type of this UIComponent subclass.

Member Summary

Constructors

`UISelectItems()`

Methods

`void encodeBegin(javax.faces.context.FacesContext context)`
Override the default behavior and perform no encoding.

`void encodeChildren(javax.faces.context.FacesContext context)`
Override the default behavior and perform no encoding.

`void encodeEnd(javax.faces.context.FacesContext context)`
Override the default behavior and perform no encoding.

`java.lang.String getComponentType()`
`boolean getRendersSelf()`
Return true to indicate that no `javax.faces.render.Renderer` needs to be associated with this component.

Inherited Member Summary

Fields inherited from interface `UIComponent`

`CLIENT_ID_ATTR`, `FACET_PARENT`, `FACET_PARENT_ATTR`, `SEPARATOR_CHAR`

Methods inherited from class `Object`

`clone()`, `equals(Object)`, `finalize()`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `toString()`, `wait()`, `wait()`

Methods inherited from class `UIComponentBase`

`addChild(int, UIComponent)`, `addChild(int, UIComponent)`, `addFacet(String, UIComponent)`, `addValidator(Validator)`, `broadcast(FacesEvent, PhaseId)`, `clearChildren()`, `clearFacets()`, `clearValidators()`, `containsChild(UIComponent)`, `currentValue(FacesContext)`, `findComponent(String)`, `getAttribute(String)`, `getAttributeNames()`, `getChild(int)`, `getChildCount()`, `getChildren()`, `getClientId(FacesContext)`, `getComponentId()`, `getConverter()`, `getFacet(String)`, `getFacetNames()`, `getFacetsAndChildren()`, `getModelReference()`, `getParent()`, `getRendererType()`, `getRendersChildren()`, `getValidators()`, `getValue()`, `isRendered()`, `isValid()`, `processDecodes(FacesContext)`, `processUpdates(FacesContext)`, `processValidators(FacesContext)`, `removeChild(UIComponent)`, `removeChild(UIComponent)`, `removeFacet(String)`, `removeValidator(Validator)`, `setAttribute(String, Object)`, `setComponentId(String)`, `setConverter(String)`, `setModelReference(String)`, `setRendered(boolean)`, `setRendererType(String)`, `setValid(boolean)`, `setValue(Object)`, `validate(FacesContext)`

Methods inherited from class `UIOutput`

Inherited Member Summary

```
decode(FacesContext)_o, updateModel(FacesContext)_o
```

Fields

TYPE

```
public static final java.lang.String TYPE
```

The component type of this `UIComponent`_o subclass.

Constructors

UISelectItems()

```
public UISelectItems()
```

Methods

encodeBegin(FacesContext)

```
public void encodeBegin(javax.faces.context.FacesContext_o context)
    throws IOException
```

Override the default behavior and perform no encoding.

Overrides: `encodeBegin`_o in class `UIComponentBase`_o

Parameters:

`context` - `javax.faces.context.FacesContext`_o for the request we are processing

Throws:

`java.io.IOException` - if an input/output error occurs while reading

`NullPointerException` - if `context` is null

encodeChildren(FacesContext)

```
public void encodeChildren(javax.faces.context.FacesContext_o context)
    throws IOException
```

Override the default behavior and perform no encoding.

Overrides: `encodeChildren`_o in class `UIComponentBase`_o

Parameters:

context - javax.faces.context.FacesContext₀ for the request we are processing

Throws:

java.io.IOException - if an input/output error occurs while reading

NullPointerException - if context is null

encodeEnd(FacesContext)

```
public void encodeEnd(javax.faces.context.FacesContext0 context)
                      throws IOException
```

Override the default behavior and perform no encoding.

Overrides: encodeEnd₀ in class UIOutput₀

Parameters:

context - javax.faces.context.FacesContext₀ for the request we are processing

Throws:

java.io.IOException - if an input/output error occurs while reading

NullPointerException - if context is null

getComponentType()

```
public java.lang.String getComponentType()
```

Description copied from interface: javax.faces.component.UIComponent₀

Return the component type of this UIComponent₀.

Overrides: getComponentType₀ in class UIOutput₀

getRendersSelf()

```
public boolean getRendersSelf()
```

Return true to indicate that no javax.faces.render.Renderer₀ needs to be associated with this component.

Overrides: getRendersSelf₀ in class UIComponentBase₀

javax.faces.component UISelectMany

Declaration

```
public class UISelectMany extends UISelectBase  
  
java.lang.Object  
|  
+--javax.faces.component.UIComponentBase0  
|  
+--javax.faces.component.UIInput0  
|  
+--javax.faces.component.UISelectBase  
|  
+--javax.faces.component.UISelectMany
```

All Implemented Interfaces: NamingContainer₀, java.io.Serializable,
UIComponent₀

Description

UISelectMany is a UIComponent₀ that represents the user's choice of a zero or more items from among a discrete set of available options. The user can modify the selected values. Optionally, the component can be preconfigured with zero or more currently selected items, by storing them as an array in the value property of the component.

This component is generally rendered as a select box or a group of checkboxes.

Member Summary

Fields

```
static TYPE0  
java.lang.String      The component type of this UIComponent0 subclass.
```

Constructors

```
UISelectMany( )0
```

Methods

```
void addComponentToNamespace(UIComponent namedComponent)  
protected boolean compareValues(java.lang.Object previous, java.lang.Object  
value)0  
                  Return true if the new value is different from the previous value.  
void decode(javax.faces.context.FacesContext context)0  
void encodeEnd(javax.faces.context.FacesContext context)0
```

Member Summary

UIComponent	findComponentInNamespace(java.lang.String name)
java.lang.String	generateClientId()
protected	getAsString(javax.faces.context.FacesContext context, java.lang.String attributeName, java.lang.String modelReference)
	Return the value of the requested attribute (or the value from the corresponding model reference expression if the attribute is not defined), as a single String.
protected	getAsStrings(javax.faces.context.FacesContext context, java.lang.String attributeName, java.lang.String modelReference)
	Return the values of the requested attribute (or the values from the corresponding model reference expression if the attribute is not defined), as an array of Strings.
java.lang.String	getComponentType()
java.lang.Object[]	getSelectedValues()
	Return the currently selected items, or null if there are no currently selected items.
protected	getSelectItems(javax.faces.context.FacesContext context)
java.util.Iterator	
	Return an Iterator over SelectItem ₀ instances representing the available options for this component, assembled from the set of UISelectItem ₀ and/or UISelectItems ₀ components that are direct children of this component.
void	removeComponentFromNamespace(UIComponent namedComponent)
void	setSelectedValues(java.lang.Object selectedItems)
	Set the currently selected items, or null to indicate that there are no currently selected items.

Inherited Member Summary

Fields inherited from interface `UIComponent0`

CLIENT_ID_ATTR₀, FACET_PARENT₀, FACET_PARENT_ATTR₀, SEPARATOR_CHAR₀

Fields inherited from class `UIInput0`

PREVIOUS_VALUE₀, listeners₀

Methods inherited from class `Object`

clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()

Methods inherited from class `UIComponentBase0`

Inherited Member Summary

```
addChild(int, UIComponent), addChild(int, UIComponent), addFacet(String, UIComponent), addValidator(Validator), clearChildren(), clearFacets(), clearValidators(), containsChild(UIComponent), currentValue(FacesContext), encodeBegin(FacesContext), encodeChildren(FacesContext), findComponent(String), getAttribute(String), getAttributeNames(), getChild(int), getChildCount(), getChildren(), getClientId(FacesContext), getComponentId(), getConverter(), getFacet(String), getFacetNames(), getFacetsAndChildren(), getModelReference(), getParent(), getRendererType(), getRendersChildren(), getRendersSelf(), getValidators(), getValue(), isRendered(), isValid(), processDecodes(FacesContext), processUpdates(FacesContext), processValidators(FacesContext), removeChild(UIComponent), removeChild(UIComponent), removeFacet(String), removeValidator(Validator), setAttribute(String, Object), setComponentId(String), setConverter(String), setModelReference(String), setRendered(boolean), setRendererType(String), setValid(boolean), setValue(Object), updateModel(FacesContext)
```

Methods inherited from class `UIInput`

```
addValueChangeListener(ValueChangedListener), broadcast(ValueChangedEvent, List), broadcast(ValueChangedEvent, List), fireValueChangeEvent(FacesContext, Object, Object), removeValueChangeListener(ValueChangedListener), validate(FacesContext)
```

Fields

TYPE

```
public static final java.lang.String TYPE
```

The component type of this `UIComponent` subclass.

Constructors

`UISelectMany()`

```
public UISelectMany()
```

Methods

`addComponentToNamespace(UIComponent)`

```
public void addComponentToNamespace(javax.faces.component.UIComponent, namedComponent)
```

Description copied from interface: javax.faces.component.NamingContainer₀

Add the specified UIComponent₀ to the namespace of this naming container.

Specified By: addComponentToNamespace₀ in interface NamingContainer₀

compareValues(Object, Object)

```
protected boolean compareValues(java.lang.Object previous, java.lang.Object value)
```

Return true if the new value is different from the previous value. Value comparison must not be sensitive to element order.

Overrides: compareValues₀ in class UIInput₀

Parameters:

previous - old value of this component

value - new value of this component

decode(FacesContext)

```
public void decode(javax.faces.context.FacesContext0 context)
                   throws IOException
```

Description copied from interface: javax.faces.component.UIComponent₀

Decode the current state of this UIComponent₀ from the request contained in the specified javax.faces.context.FacesContext₀, and attempt to convert this state information into an object of the required type for this component (optionally using the registered javax.faces.convert.Converter₀ for this component, if there is one).

If conversion is successful:

- Save the new local value of this component by calling setValue() and passing the new value.
- Set the valid property of this component to true.

If conversion is not successful:

- Save state information in such a way that encoding can reproduce the previous input (even though it was syntactically or semantically incorrect)
- Add an appropriate conversion failure error message by calling context.addMessage().
- Set the valid property of this component to false.

During decoding, events may be enqueued for later processing (by event listeners who have registered an interest), by calling addFacesEvent() on the associated javax.faces.context.FacesContext₀.

Overrides: decode₀ in class UIInput₀

Throws:

java.io.IOException

encodeEnd(FacesContext)

```
public void encodeEnd(javax.faces.context.FacesContext0 context)
                      throws IOException
```

Description copied from interface: javax.faces.component.UIComponent₀

Render the ending of the current state of this UIComponent₀, following the rules described for encodeBegin() to acquire the appropriate value to be rendered.

Overrides: encodeEnd₀ in class UIInput₀

Throws:

java.io.IOException

findComponentInNamespace(String)

```
public javax.faces.component.UIComponent0
        findComponentInNamespace(java.lang.String name)
```

Description copied from interface: javax.faces.component.NamingContainer₀

Find and return a UIComponent₀ in this namespace, if it is present; otherwise return null.

If the argument name does not contain any UIComponent. SEPARATOR_CHAR₀ characters, it is interpreted to be a name in the namespace of this naming container.

If the argument name does contain UIComponent. SEPARATOR_CHAR₀ characters, each segment between UIComponent. SEPARATOR_CHAR₀ is treated as a component identifier in its own namespace, which are searched for in child naming containers of this naming container, from left to right.

Specified By: findComponentInNamespace₀ in interface NamingContainer₀

generateClientId()

```
public java.lang.String generateClientId()
```

Description copied from interface: javax.faces.component.NamingContainer₀

Generate an identifier for a component, suitable for communication to a client. The identifier returned from this method must be unique within this namespace.

Specified By: generateClientId₀ in interface NamingContainer₀

getAsString(FacesContext, String, String)

```
protected java.lang.String getAsString(javax.faces.context.FacesContext0 context,  
                                     java.lang.String attributeName, java.lang.String modelReference)
```

Return the value of the requested attribute (or the value from the corresponding model reference expression if the attribute is not defined), as a single String. If no data can be acquired, return a zero-length String.

Parameters:

- context - The javax.faces.context.FacesContext₀ for the current request
- attributeName - Attribute name of the attribute defining the local value of the requested data (if any)
- modelReference - Model reference expression defining the external source of the requested data (if any)

getAsStrings(FacesContext, String, String)

```
protected java.lang.String[] getAsStrings(javax.faces.context.FacesContext0  
                                         context, java.lang.String attributeName,  
                                         java.lang.String modelReference)
```

Return the values of the requested attribute (or the values from the corresponding model reference expression if the attribute is not defined), as an array of Strings. If no data can be acquired, return a zero-length array of Strings.

Parameters:

- context - The javax.faces.context.FacesContext₀ for the current request
- attributeName - Attribute name for the attribute defining the local values of the requested data (if any)
- modelReference - Model reference expression defining the external source of the requested data (if any)

getComponentType()

```
public java.lang.String getComponentType()
```

Description copied from interface: javax.faces.component.UIComponent₀

Return the component type of this UIComponent₀.

Overrides: getComponentType₀ in class UIInput₀

getSelectedValues()

```
public java.lang.Object[] getSelectedValues()
```

Return the currently selected items, or null if there are no currently selected items.

getSelectItems(FacesContext)

```
protected java.util.Iterator getSelectItems(javax.faces.context.FacesContext context)
```

Return an Iterator over SelectItem₀ instances representing the available options for this component, assembled from the set of UISelectItem₀ and/or UISelectItems₀ components that are direct children of this component. If there are no such children, a zero-length array is returned.

Parameters:

context - The javax.faces.context.FacesContext₀ for the current request

Throws:

java.lang.IllegalArgumentException - if the value of a UISelectItem₀ or UISelectItems₀ is of the wrong type

NullPointerException - if context is null

removeComponentFromNamespace(UIComponent)

```
public void removeComponentFromNamespace(javax.faces.component.UIComponent namedComponent)
```

Description copied from interface: javax.faces.component.NamingContainer₀

Remove the specified UIComponent₀ from the namespace of this naming container, if it is present.

Specified By: removeComponentFromNamespace₀ in interface NamingContainer₀

setSelectedValues(Object[])

```
public void setSelectedValues(java.lang.Object[] selectedItems)
```

Set the currently selected items, or null to indicate that there are no currently selected items.

Parameters:

selectedItemss - The new selected items (if any)

javax.faces.component UISelectOne

Declaration

```
public class UISelectOne extends UISelectBase  
  
java.lang.Object  
|  
+--javax.faces.component.UIComponentBase0  
|  
+--javax.faces.component.UIInput0  
|  
+--javax.faces.component.UISelectBase  
|  
+--javax.faces.component.UISelectOne
```

All Implemented Interfaces: NamingContainer₀, java.io.Serializable,
UIComponent₀

Description

UISelectOne is a UIComponent₀ that represents the user's choice of a single item from among a discrete set of available options. The user can modify the selected value. Optionally, the component can be preconfigured with a currently selected item, by storing it as the value property of the component.

This component is generally rendered as a select box or a group of radio buttons.

Member Summary

Fields

static TYPE ₀	
java.lang.String	The component type of this UIComponent ₀ subclass.

Constructors

UISelectOne()	
---------------	--

Methods

void	addComponentToNamespace(UIComponent namedComponent)
void	decode(javax.faces.context.FacesContext context)
void	encodeEnd(javax.faces.context.FacesContext context)
UIComponent	findComponentInNamespace(java.lang.String name)
java.lang.String	generateClientId()

Member Summary

protected	getAsString(javax.faces.context.FacesContext context, java.lang.String attributeName, java.lang.String modelReference)	Return the value of the requested attribute (or the value from the corresponding model reference expression if the attribute is not defined), as a single String.
protected	getAsStrings(javax.faces.context.FacesContext context, java.lang.String[] attributeName, java.lang.String modelReference)	Return the values of the requested attribute (or the values from the corresponding model reference expression if the attribute is not defined), as an array of Strings.
java.lang.String	getComponentType()	
java.lang.Object	getSelectedValue()	Return the currently selected item, or null if there is no currently selected item.
protected	getSelectItems(javax.faces.context.FacesContext context)	
java.util.Iterator		Return an Iterator over SelectItem ₀ instances representing the available options for this component, assembled from the set of UISelectItem ₀ and/or UISelectItems ₀ components that are direct children of this component.
void	removeComponentFromNamespace(UIComponent namedComponent)	
void	setSelectedValue(java.lang.Object selectedItem)	Set the currently selected item, or null to indicate that there is no currently selected item.

Inherited Member Summary**Fields inherited from interface `UIComponent0`**CLIENT_ID_ATTR₀, FACET_PARENT₀, FACET_PARENT_ATTR₀, SEPARATOR_CHAR₀**Fields inherited from class `UIInput0`**PREVIOUS_VALUE₀, listeners₀**Methods inherited from class `Object`**

clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()

Methods inherited from class `UIComponentBase0`

Inherited Member Summary

```
addChild(int, UIComponent)o, addChild(int, UIComponent)o, addFacet(String, UIComponent)o, addValidator(Validator)o, clearChildren()o, clearFacets()o, clearValidators()o, containsChild(UIComponent)o, currentValue(FacesContext)o, encodeBegin(FacesContext)o, encodeChildren(FacesContext)o, findComponent(String)o, getAttribute(String)o, getAttributeNames()o, getChild(int)o, getChildCount()o, getChildren()o, getClientId(FacesContext)o, getComponentId()o, getConverter()o, getFacet(String)o, getFacetNames()o, getFacetsAndChildren()o, getModelReference()o, getParent()o, getRendererType()o, getRendersChildren()o, getRendersSelf()o, getValidators()o, getValue()o, isRendered()o, isValid()o, processDecodes(FacesContext)o, processUpdates(FacesContext)o, processValidators(FacesContext)o, removeChild(UIComponent)o, removeChild(UIComponent)o, removeFacet(String)o, removeValidator(Validator)o, setAttribute(String, Object)o, setComponentId(String)o, setConverter(String)o, setModelReference(String)o, setRendered(boolean)o, setRendererType(String)o, setValid(boolean)o, setValue(Object)o, updateModel(FacesContext)o
```

Methods inherited from class `UIInput`,

```
addValueChangeListener(ValueChangedListener)o, broadcast(ValueChangedEvent, List)o, broadcast(ValueChangedEvent, List)o, compareValues(Object, Object)o, fireValueChangeEvent(FacesContext, Object, Object)o, removeValueChangeListener(ValueChangedListener)o, validate(FacesContext)o
```

Fields

`TYPE`

```
public static final java.lang.String TYPE
```

The component type of this `UIComponent` subclass.

Constructors

`UISelectOne()`

```
public UISelectOne()
```

Methods

`addComponentToNamespace(UIComponent)`

```
public void addComponentToNamespace(javax.faces.component.UIComponent o, namedComponent)
```

Description copied from interface: `javax.faces.component.NamingContainer`

Add the specified `UIComponent` to the namespace of this naming container.

Specified By: `addComponentToNamespace` in interface `NamingContainer`

`decode(FacesContext)`

```
public void decode(javax.faces.context.FacesContext context)
                  throws IOException
```

Description copied from interface: `javax.faces.component.UIComponent`

Decode the current state of this `UIComponent` from the request contained in the specified `javax.faces.context.FacesContext`, and attempt to convert this state information into an object of the required type for this component (optionally using the registered `javax.faces.convert.Converter` for this component, if there is one).

If conversion is successful:

- Save the new local value of this component by calling `setValue()` and passing the new value.
- Set the `valid` property of this component to `true`.

If conversion is not successful:

- Save state information in such a way that encoding can reproduce the previous input (even though it was syntactically or semantically incorrect)
- Add an appropriate conversion failure error message by calling `context.addMessage()`.
- Set the `valid` property of this component to `false`.

During decoding, events may be enqueued for later processing (by event listeners who have registered an interest), by calling `addFacesEvent()` on the associated `javax.faces.context.FacesContext`.

Overrides: `decode` in class `UIInput`

Throws:

`java.io.IOException`

`encodeEnd(FacesContext)`

```
public void encodeEnd(javax.faces.context.FacesContext context)
                      throws IOException
```

Description copied from interface: javax.faces.component.UIComponent₀

Render the ending of the current state of this UIComponent₀, following the rules described for encodeBegin() to acquire the appropriate value to be rendered.

Overrides: encodeEnd₀ in class UIInput₀

Throws:

java.io.IOException

findComponentInNamespace(String)

```
public javax.faces.component.UIComponent0
    findComponentInNamespace(java.lang.String name)
```

Description copied from interface: javax.faces.component.NamingContainer₀

Find and return a UIComponent₀ in this namespace, if it is present; otherwise return null.

If the argument name does not contain any UIComponent . SEPARATOR_CHAR₀ characters, it is interpreted to be a name in the namespace of this naming container.

If the argument name does contain UIComponent . SEPARATOR_CHAR₀ characters, each segment between UIComponent . SEPARATOR_CHAR₀ is treated as a component identifier in its own namespace, which are searched for in child naming containers of this naming container, from left to right.

Specified By: findComponentInNamespace₀ in interface NamingContainer₀

generateClientId()

```
public java.lang.String generateClientId()
```

Description copied from interface: javax.faces.component.NamingContainer₀

Generate an identifier for a component, suitable for communication to a client. The identifier returned from this method must be unique within this namespace.

Specified By: generateClientId₀ in interface NamingContainer₀

getAsString(FacesContext, String, String)

```
protected java.lang.String getAsString(javax.faces.context.FacesContext0 context,
    java.lang.String attributeName, java.lang.String modelReference)
```

Return the value of the requested attribute (or the value from the corresponding model reference expression if the attribute is not defined), as a single String. If no data can be acquired, return a zero-length String.

Parameters:

context - The javax.faces.context.FacesContext₀ for the current request
attributeName - Attribute name of the attribute defining the local value of the requested data (if any)
modelReference - Model reference expression defining the external source of the requested data (if any)

getAsStrings(FacesContext, String, String)

```
protected java.lang.String[] getAsStrings(javax.faces.context.FacesContext0
    context, java.lang.String attributeName,
    java.lang.String modelReference)
```

Return the values of the requested attribute (or the values from the corresponding model reference expression if the attribute is not defined), as an array of Strings. If no data can be acquired, return a zero-length array of Strings.

Parameters:

context - The javax.faces.context.FacesContext₀ for the current request
attributeName - Attribute name for the attribute defining the local values of the requested data (if any)
modelReference - Model reference expression defining the external source of the requested data (if any)

getComponentType()

```
public java.lang.String getComponentType()
```

Description copied from interface: javax.faces.component.UIComponent₀

Return the component type of this UIComponent₀.

Overrides: getComponentType₀ in class UIInput₀

getSelectedValue()

```
public java.lang.Object getSelectedValue()
```

Return the currently selected item, or null if there is no currently selected item.

getSelectItems(FacesContext)

```
protected java.util.Iterator getSelectItems(javax.faces.context.FacesContext0
    context)
```

Return an Iterator over `SelectItem0` instances representing the available options for this component, assembled from the set of `UISelectItem0` and/or `UISelectItems0` components that are direct children of this component. If there are no such children, a zero-length array is returned.

Parameters:

`context` - The `javax.faces.context.FacesContext0` for the current request

Throws:

`java.lang.IllegalArgumentException` - if the value of a `UISelectItem0` or `UISelectItems0` is of the wrong type

`NullPointerException` - if `context` is null

removeComponentFromNamespace(UIComponent)

```
public void removeComponentFromNamespace(javax.faces.component.UIComponent0
                                         namedComponent)
```

Description copied from interface: `javax.faces.component.NamingContainer0`

Remove the specified `UIComponent0` from the namespace of this naming container, if it is present.

Specified By: `removeComponentFromNamespace0` in interface `NamingContainer0`

setSelectedValue(Object)

```
public void setSelectedValue(java.lang.Object selectedItem)
```

Set the currently selected item, or `null` to indicate that there is no currently selected item.

Parameters:

`selectedItem` - The new selected item (if any)

JSF.10.3 Package `javax.faces.context`

Package `javax.faces.context`

Description

Classes and interfaces defining per-request state information.

Class Summary

Interfaces

`Message0`

Message represents a single validation (or other) message, which is typically associated with a particular component in the component tree.

Classes

`FacesContext0`

FacesContext contains all of the per-request state information related to the processing of a single JavaServer Faces request, and the rendering of the corresponding response.

`FacesContextFactory0`

FacesContextFactory is a factory object that creates (if needed) and returns new `FacesContext0` instances, initialized for the processing of the specified request and response objects.

`MessageImpl0`

MessageImpl is a concrete class that implements `Message0`, and is a convenient base class for other `Message0` implementations.

`MessageResources0`

MessageResources represents a collection of message templates, uniquely identified by message identifiers, that can be used to construct localized `Message0` instances to be added to the message list in a `FacesContext0`.

`MessageResourcesFactory0`

MessageResourcesFactory is a Factory object that creates (if needed) and returns `MessageResources0` instances.

`ResponseStream0`

ResponseStream is an interface describing an adapter to an underlying output mechanism for binary output.

`ResponseWriter0`

ResponseWriter is an abstract class describing an adapter to an underlying output mechanism for character-based output.

javax.faces.context FacesContext

Declaration

```
public abstract class FacesContext  
  
java.lang.Object  
|  
+--javax.faces.context.FacesContext
```

Description

FacesContext contains all of the per-request state information related to the processing of a single JavaServer Faces request, and the rendering of the corresponding response. It is passed to, and potentially modified by, each phase of the request processing lifecycle.

A `FacesContext0` instance is associated with a particular request at the beginning of request processing, by a call to the `getFacesContext()` method of the `FacesContextFactory0` instance associated with the current web application. The instance remains active until its `release()` method is called, after which no further references to this instance are allowed. While a `FacesContext0` instance is active, it must not be referenced from any thread other than the one upon which the servlet container executing this web application utilizes for the processing of this request.

Member Summary

Fields

<code>static FACES_CONTEXT_ATTR₀ java.lang.String</code>	The name of the request attribute under which the <code>FacesContext₀</code> instance for the current request will be stored.
---	--

Constructors

`FacesContext()0`

Methods

<code>abstract void addApplicationEvent(javax.faces.event.ApplicationEvent event)₀</code>	Append a <code>javax.faces.event.ApplicationEvent₀</code> to the set of events that should be processed by the application during the <i>Invoke Application</i> phase of the request processing lifecycle.
<code>abstract void addFacesEvent(javax.faces.event.FacesEvent event)₀</code>	Append a <code>javax.faces.event.FacesEvent₀</code> to the set of events that should be processed at the end of the current phase of the request processing lifecycle.

Member Summary

abstract void addMessage(javax.faces.component.UIComponent component, Message message)	Append a Message _o to the set of messages associated with the specified javax.faces.component.UIComponent _o , if component is not null.
abstract java.util.Iterator getApplicationEvents()	Return an Iterator over the javax.faces.event.ApplicationEvent _o s for events that should be handled by the application during the <i>Invoke Application</i> phase of the request processing lifecycle.
abstract int getApplicationEventsCount()	Return the number of javax.faces.event.ApplicationEvent _o s that have been queued to the application, or zero if no such events have been queued.
abstract javax.faces.lifecycle.ApplicationHandler getApplicationHandler()	Return the javax.faces.lifecycle.ApplicationHandler _o instance to be utilized during the <i>Invoke Application</i> phase of the request processing lifecycle.
static FacesContext getCurrentInstance()	Return the FacesContext _o instance for the request that is being processed by the current thread.
abstract java.util.Iterator getFacesEvents()	Return an Iterator over the javax.faces.event.FacesEvent _o s that have been queued, in the order that they were queued.
abstract javax.servlet.http.HttpSession getSession()	Return the HttpSession instance for the session associated with the current request (if any); otherwise, return null.
abstract java.util.Locale getLocale()	Return the Locale to be used in localizing the response being created for this FacesContext.
abstract int getMaximumSeverity()	Return the maximum severity level recorded on any Message _o s that has been queued, whether or not they are associated with any specific javax.faces.component.UIComponent _o .
abstract java.util.Iterator getMessages()	Return an Iterator over the Message _o s that have been queued, whether or not they are associated with any specific javax.faces.component.UIComponent _o .
abstract java.util.Iterator getMessages(javax.faces.component.UIComponent component)	Return an Iterator over the Message _o s that have been queued that are associated with the specified javax.faces.component.UIComponent _o , (if component is not null), or over the Message _o s that have been queued that are not associated with any specific javax.faces.component.UIComponent _o (if component is null).
abstract java.lang.Class getModelType(java.lang.String modelReference)	Evaluate the specified model reference expression, and return the expected type of the corresponding value, if it can be determined; otherwise, return null.
abstract java.lang.Object getModelValue(java.lang.String modelReference)	Evaluate the specified model reference expression, and return the corresponding data value (which may be null).

Member Summary

abstract	getResponseStream()	
ResponseStream		Return the ResponseStream ₀ to which components should direct their binary output.
abstract	getResponseWriter()	
ResponseWriter		Return the ResponseWriter ₀ to which components should direct their character-based output.
abstract	getServletContext()	
javax.servlet.ServletContext		Return the ServletContext object for the web application associated with this request.
abstract	getServletRequest()	
javax.servlet.ServletRequest		Return the ServletRequest object representing the current request that is being processed.
abstract	getServletResponse()	
javax.servlet.ServletResponse		Return the ServletResponse object representing the current response that is being rendered.
abstract	getTree()	
javax.faces.tree.Tree		Return the component javax.faces.tree.Tree ₀ that is associated with the this request.
abstract	getViewHandler()	
javax.faces.lifecycle.ViewHandler		Return the javax.faces.lifecycle.ViewHandler ₀ instance to be utilized during the <i>Render Response</i> phase of the request processing lifecycle.
abstract void	release()	
		Release any resources associated with this FacesContext instance.
abstract void	renderResponse()	
		Signal the JavaServer faces implementation that, as soon as the current phase of the request processing lifecycle has been completed, control should be passed to the <i>Render Response</i> phase, bypassing any phases that have not been executed yet.
abstract void	responseComplete()	
		Signal the JavaServer Faces implementation that the HTTP response for this request has already been generated (such as an HTTP redirect), and that the request processing lifecycle should be terminated as soon as the current phase is completed.
protected static void	setCurrentInstance(FacesContext context)	
		Set the FacesContext ₀ instance for the request that is being processed by the current thread.
abstract void	setLocale(java.util.Locale locale)	
		Set the Locale to be used in localizing the response being created for this FacesContext.
abstract void	setModelValue(java.lang.String modelReference, java.lang.Object value)	
		Evaluate the specified model reference expression, and set the corresponding data value (which may be null).
abstract void	setResponseStream(ResponseStream responseStream)	
		Set the ResponseStream ₀ to which components should direct their binary output.
abstract void	setResponseWriter(ResponseWriter responseWriter)	
		Set the ResponseWriter ₀ to which components should direct their character-based output.

Member Summary

```
abstract void setTree(javax.faces.tree.Tree tree)
```

Set the component `javax.faces.tree.Tree` that is associated with this request.

Inherited Member Summary

Methods inherited from class `Object`

```
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(),
toString(), wait(), wait(), wait()
```

Fields

`FACES_CONTEXT_ATTR`

```
public static final java.lang.String FACES_CONTEXT_ATTR
```

The name of the request attribute under which the `FacesContext` instance for the current request will be stored.

Constructors

`FacesContext()`

```
public FacesContext()
```

Methods

`addApplicationEvent(ApplicationEvent)`

```
public abstract void addApplicationEvent(javax.faces.event.ApplicationEvent,
                                         event)
```

Deprecated. The current mechanism for handling application events is a placeholder, and will be replaced in the next public release of JavaServer Faces.

Append a `javax.faces.event.ApplicationEvent` to the set of events that should be processed by the application during the *Invoke Application* phase of the request processing lifecycle.

Parameters:

`event` - The event to be appended

Throws:

NullPointerException - if event is null

addFacesEvent(FacesEvent)

```
public abstract void addFacesEvent(javax.faces.event.FacesEvent0 event)
```

Append a javax.faces.event.FacesEvent₀ to the set of events that should be processed at the end of the current phase of the request processing lifecycle.

Parameters:

event - The event to be queued

Throws:

NullPointerException - if event is null

addMessage(UIComponent, Message)

```
public abstract void addMessage(javax.faces.component.UIComponent0 component,  
                                javax.faces.context.Message0 message)
```

Append a Message₀ to the set of messages associated with the specified javax.faces.component.UIComponent₀, if component is not null. If component is null, this Message₀ is assumed to not be associated with any specific component instance.

Parameters:

component - The component with which this message is associated (if any)

message - The message to be appended

Throws:

NullPointerException - if message is null

getApplicationEvents()

```
public abstract java.util.Iterator getApplicationEvents()
```

Deprecated. The current mechanism for handling application events is a placeholder, and will be replaced in the next public release of JavaServer Faces.

Return an Iterator over the javax.faces.event.ApplicationEvent₀s for events that should be handled by the application during the *Invoke Application* phase of the request processing lifecycle.

getApplicationEventsCount()

```
public abstract int getApplicationEventsCount()
```

Deprecated. The current mechanism for handling application events is a placeholder, and will be replaced in the next public release of JavaServer Faces.

Return the number of `javax.faces.event.ApplicationEvent`s that have been queued to the application, or zero if no such events have been queued.

getApplicationHandler()

```
public abstract javax.faces.lifecycle.ApplicationHandler getApplicationHandler()
```

Deprecated. The current mechanism for handling application events is a placeholder, and will be replaced in the next public release of JavaServer Faces.

Return the `javax.faces.lifecycle.ApplicationHandler` instance to be utilized during the *Invoke Application* phase of the request processing lifecycle.

getCurrentInstance()

```
public static javax.faces.context.FacesContext getCurrentInstance()
```

Return the `FacesContext` instance for the request that is being processed by the current thread.

getFacesEvents()

```
public abstract java.util.Iterator getFacesEvents()
```

Return an `Iterator` over the `javax.faces.event.FacesEvent`s that have been queued, in the order that they were queued. The `Iterator` returned by this method must support the following functionality:

- The `remove()` operation may be utilized by the JSF implementation to remove the current event when it has been completely handled.
- Event handlers may add new events to the list while it is being iterated over, by calling `addFacesEvent()`.

If no events have been queued, an empty `Iterator` must be returned.

getHttpSession()

```
public abstract javax.servlet.http.HttpSession getHttpSession()
```

Return the `HttpSession` instance for the session associated with the current request (if any); otherwise, return null.

getLocale()

```
public abstract java.util.Locale getLocale()
```

Return the `Locale` to be used in localizing the response being created for this `FacesContext`.

getMaximumSeverity()

```
public abstract int getMaximumSeverity()
```

Return the maximum severity level recorded on any `Message`s that has been queued, whether or not they are associated with any specific `javax.faces.component.UIComponent`. If no such messages have been queued, return a value less than `Message.SEVERITY_INFO`.

getMessages()

```
public abstract java.util.Iterator getMessages()
```

Return an Iterator over the `Message`s that have been queued, whether or not they are associated with any specific `javax.faces.component.UIComponent`. If no such messages have been queued, return an empty Iterator.

getMessages(UIComponent)

```
public abstract java.util.Iterator getMessages( javax.faces.component.UIComponent component )
```

Return an Iterator over the `Message`s that have been queued that are associated with the specified `javax.faces.component.UIComponent`, (if component is not null), or over the `Message`s that have been queued that are not associated with any specific `javax.faces.component.UIComponent` (if component is null). If no such messages have been queued, return an empty Iterator.

Parameters:

`component` - The `javax.faces.component.UIComponent` for which messages are requested, or `null` for messages not associated with any component

getModelType(String)

```
public abstract java.lang.Class getModelType( java.lang.String modelReference )  
throws FacesException
```

Evaluate the specified model reference expression, and return the expected type of the corresponding value, if it can be determined; otherwise, return `null`.

Parameters:

`modelReference` - Model reference expression to be evaluated

Throws:

`javax.faces.FacesException` - if an error occurs during expression evaluation

`java.lang.IllegalArgumentException` - if the model reference expression is invalid

`NullPointerException` - if `modelReference` is `null`

getModelValue(String)

```
public abstract java.lang.Object getModelValue(java.lang.String modelReference)  
throws FacesException
```

Evaluate the specified model reference expression, and return the corresponding data value (which may be null). No data type conversion is performed.

Parameters:

modelReference - Model reference to be evaluated

Throws:

javax.faces.FacesException₀ - if an error occurs during expression evaluation

java.lang.IllegalArgumentException - if the model reference expression is invalid

NullPointerException - if modelReference is null

getResponseStream()

```
public abstract javax.faces.context.ResponseStream0 getResponseStream( )
```

Return the ResponseStream₀ to which components should direct their binary output. Within a given response, components can use either the ResponseStream or the ResponseWriter, but not both.

getResponseWriter()

```
public abstract javax.faces.context.ResponseWriter0 getResponseWriter( )
```

Return the ResponseWriter₀ to which components should direct their character-based output. Within a given response, components can use either the ResponseStream or the ResponseWriter, but not both.

getServletContext()

```
public abstract javax.servlet.ServletContext getServletContext( )
```

Return the ServletContext object for the web application associated with this request.

getServletRequest()

```
public abstract javax.servlet.ServletRequest getServletRequest( )
```

Return the ServletRequest object representing the current request that is being processed.

getServletResponse()

```
public abstract javax.servlet.ServletResponse getServletResponse( )
```

Return the ServletResponse object representing the current response that is being rendered.

getTree()

```
public abstract javax.faces.tree.Tree0 getTree()
```

Return the component `javax.faces.tree.Tree0` that is associated with the this request.

getViewHandler()

```
public abstract javax.faces.lifecycle.ViewHandler0 getViewHandler()
```

Return the `javax.faces.lifecycle.ViewHandler0` instance to be utilized during the *Render Response* phase of the request processing lifecycle.

release()

```
public abstract void release()
```

Release any resources associated with this `FacesContext` instance. Faces implementations may choose to pool instances in the associated `FacesContextFactory0` to avoid repeated object creation and garbage collection.

renderResponse()

```
public abstract void renderResponse()
```

Signal the JavaSerer faces implementation that, as soon as the current phase of the request processing lifecycle has been completed, control should be passed to the *Render Response* phase, bypassing any phases that have not been executed yet.

responseComplete()

```
public abstract void responseComplete()
```

Signal the JavaServer Faces implementation that the HTTP response for this request has already been generated (such as an HTTP redirect), and that the request processing lifecycle should be terminated as soon as the current phase is completed.

setCurrentInstance(FacesContext)

```
protected static void setCurrentInstance(javax.faces.context.FacesContext0
                                         context)
```

Set the `FacesContext0` instance for the request that is being processed by the current thread.

Parameters:

context - The `FacesContext0` instance for the current thread

setLocale(Locale)

```
public abstract void setLocale(java.util.Locale locale)
```

Set the Locale to be used in localizing the response being created for this FacesContext. If not set, the default Locale for our servlet container will be used.

Parameters:

locale - The new localization Locale

setModelValue(String, Object)

```
public abstract void setModelValue(java.lang.String modelReference,
                                  java.lang.Object value)
                                  throws FacesException
```

Evaluate the specified model reference expression, and set the corresponding data value (which may be null). No data type conversion is performed.

Parameters:

modelReference - Model reference to be evaluated

value - New model data to be stored in the model

Throws:

javax.faces.FacesException₀ - if an error occurs during expression evaluation

java.lang.IllegalArgumentException - if the model reference expression is invalid

NullPointerException - if modelReference is null

setResponseStream(ResponseStream)

```
public abstract void setResponseStream(javax.faces.context.ResponseStream,
                                         responseStream)
```

Set the ResponseStream₀ to which components should direct their binary output.

Parameters:

responseStream - The new ResponseStream for this response

Throws:

NullPointerException - if responseStream is null

setResponseWriter(ResponseWriter)

```
public abstract void setResponseWriter(javax.faces.context.ResponseWriter,
                                         responseWriter)
```

Set the ResponseWriter₀ to which components should direct their character-based output.

Parameters:

`responseWriter` - The new ResponseWriter for this response

Throws:

`NullPointerException` - if `responseWriter` is null

setTree(Tree)

```
public abstract void setTree(javax.faces.tree.Tree0 tree)
```

Set the component `javax.faces.tree.Tree0` that is associated with this request. This method can only be called by the application handler (or a class that the handler calls), and only during the *Invoke Application* phase of the request processing lifecycle.

Parameters:

`tree` - The new component tree

Throws:

`java.lang.IllegalStateException` - if this method is called more than once without a call to `release()` in between

`NullPointerException` - if `tree` is null

javax.faces.context

FacesContextFactory

Declaration

```
public abstract class FacesContextFactory  
  
java.lang.Object  
|  
+--javax.faces.context.FacesContextFactory
```

Description

FacesContextFactory is a factory object that creates (if needed) and returns new `FacesContext` instances, initialized for the processing of the specified request and response objects. Implementations may take advantage of the calls to the `release()` method of the allocated `FacesContext` instances to pool and recycle them, rather than creating a new instance every time.

There must be one `FacesContextFactory` instance per web application that is utilizing JavaServer Faces. This instance can be acquired, in a portable manner, by calling:

```
FacesContextFactory factory = (FacesContextFactory)  
    FactoryFinder.getFactory(FactoryFinder.FACES_CONTEXT_FACTORY);
```

Member Summary

Constructors

```
FacesContextFactory()
```

Methods

```
abstract FacesContext getFacesContext(javax.servlet.ServletContext context,  
                                      javax.servlet.ServletRequest request,  
                                      javax.servlet.ServletResponse response,  
                                      javax.faces.lifecycle.Lifecycle lifecycle)
```

Create (if needed) and return a `FacesContext` instance that is initialized for the processing of the specified request and response objects, utilizing the specified `javax.faces.lifecycle.Lifecycle` instance, for this web application.

Inherited Member Summary

Methods inherited from class Object

Inherited Member Summary

```
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(),
toString(), wait(), wait(), wait()
```

Constructors

FacesContextFactory()

```
public FacesContextFactory()
```

Methods

getFacesContext(ServletContext, ServletRequest, ServletResponse, Lifecycle)

```
public abstract javax.faces.context.FacesContext_
    getFacesContext(javax.servlet.ServletContext context,
                    javax.servlet.ServletRequest request,
                    javax.servlet.ServletResponse response,
                    javax.faces.lifecycle.Lifecycle_ lifecycle)
    throws FacesException
```

Create (if needed) and return a `FacesContext0` instance that is initialized for the processing of the specified request and response objects, utilizing the specified `javax.faces.lifecycle.Lifecycle0` instance, for this web application.

Parameters:

`context` - The `ServletContext` that is associated with this web application

`request` - The `ServletRequest` that is to be processed

`response` - The `ServletResponse` that is to be created

`lifecycle` - The `javax.faces.lifecycle.Lifecycle0` instance being used to process this request

Throws:

`javax.faces.FacesException0` - if a `FacesContext0` cannot be constructed for the specified parameters

`NullPointerException` - if any of the parameters are null

javax.faces.context Message

Declaration

```
public interface Message
```

All Known Implementing Classes: `MessageImpl0`

Description

Message represents a single validation (or other) message, which is typically associated with a particular component in the component tree.

Member Summary

Fields

```
static int SEVERITY_ERROR0
    Message severity level indicating that an error has occurred.
static int SEVERITY_FATAL0
    Message severity level indicating that a serious error has occurred.
static int SEVERITY_INFO0
    Message severity level indicating an informational message rather than an error.
static int SEVERITY_WARN0
    Message severity level indicating that an error might have occurred.
```

Methods

```
java.lang.String getDetail()0
    Return the localized detail text for this Message0 (if any).
int getSeverity()0
    Return the severity level of this Message0.
java.lang.String getSummary()0
    Return the localized summary text for this Message0.
```

Fields

SEVERITY_ERROR

```
public static final int SEVERITY_ERROR
```

Message severity level indicating that an error has occurred.

SEVERITY_FATAL

```
public static final int SEVERITY_FATAL
```

Message severity level indicating that a serious error has occurred.

SEVERITY_INFO

```
public static final int SEVERITY_INFO
```

Message severity level indicating an informational message rather than an error.

SEVERITY_WARN

```
public static final int SEVERITY_WARN
```

Message severity level indicating that an error might have occurred.

Methods

getDetail()

```
public java.lang.String getDetail()
```

Return the localized detail text for this `Message0` (if any). This should be additional text that can help the user understand the context of the problem, and offer suggestions for approaches to correcting it.

getSeverity()

```
public int getSeverity()
```

Return the severity level of this `Message0`.

getSummary()

```
public java.lang.String getSummary()
```

Return the localized summary text for this `Message0`. This should be the text that would normally be displayed to a user to identify the problem that needs to be corrected.

javax.faces.context MessageImpl

Declaration

```
public class MessageImpl implements Message0, java.io.Serializable  
  
java.lang.Object  
|  
+-- javax.faces.context.MessageImpl
```

All Implemented Interfaces: Message₀, java.io.Serializable

Description

MessageImpl is a concrete class that implements Message₀, and is a convenient base class for other Message₀ implementations.

Member Summary

Constructors

```
MessageImpl()0  
    Construct a new Message0 with no initial values.  
MessageImpl(int severity, java.lang.String summary,  
           java.lang.String detail)0  
    Construct a new Message0 with the specified initial values.
```

Methods

```
java.lang.String getDetail()0  
        int getSeverity()0  
java.lang.String getSummary()0  
        void setDetail(java.lang.String detail)0  
            Set the localized detail text.  
        void setSeverity(int severity)0  
            Set the severity level.  
        void setSummary(java.lang.String summary)0  
            Set the localized summary text.
```

Inherited Member Summary

Fields inherited from interface `Message0`

`SEVERITY_ERROR0, SEVERITY_FATAL0, SEVERITY_INFO0, SEVERITY_WARN0`

Methods inherited from class `Object`

`clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(),
toString(), wait(), wait(), wait()`

Constructors

`MessageImpl()`

```
public MessageImpl()
```

Construct a new `Message0` with no initial values.

`MessageImpl(int, String, String)`

```
public MessageImpl(int severity, java.lang.String summary,  
                   java.lang.String detail)
```

Construct a new `Message0` with the specified initial values.

Parameters:

`severity` - Severity level of this `Message0`

`summary` - Localized summary message text

`detail` - Localized detail message text

Throws:

`java.lang.IllegalArgumentException` - if the specified severity level is not one of the supported values

Methods

`getDetail()`

```
public java.lang.String getDetail()
```

Description copied from interface: javax.faces.context.Message₀

Return the localized detail text for this Message₀ (if any). This should be additional text that can help the user understand the context of the problem, and offer suggestions for approaches to correcting it.

Specified By: getDetail₀ in interface Message₀

getSeverity()

```
public int getSeverity()
```

Description copied from interface: javax.faces.context.Message₀

Return the severity level of this Message₀.

Specified By: getSeverity₀ in interface Message₀

getSummary()

```
public java.lang.String getSummary()
```

Description copied from interface: javax.faces.context.Message₀

Return the localized summary text for this Message₀. This should be the text that would normally be displayed to a user to identify the problem that needs to be corrected.

Specified By: getSummary₀ in interface Message₀

setDetail(String)

```
public void setDetail(java.lang.String detail)
```

Set the localized detail text.

Parameters:

detail - The new localized detail text

setSeverity(int)

```
public void setSeverity(int severity)
```

Set the severity level.

Parameters:

severity - The new severity level

Throws:

java.lang.IllegalArgumentException - if the specified severity level is not one of the supported values

setSummary(String)

```
public void setSummary(java.lang.String summary)
```

Set the localized summary text.

Parameters:

summary - The new localized summary text

javax.faces.context MessageResources

Declaration

```
public abstract class MessageResources  
  
java.lang.Object  
|  
+--javax.faces.context.MessageResources
```

Description

MessageResources represents a collection of message templates, uniquely identified by message identifiers, that can be used to construct localized `Message0` instances to be added to the message list in a `FacesContext0`.

An instance of `MessageResources` is created by calling the `getMessageResources()` method of `MessageResourcesFactory0`, for a specified message resources identifier. Because this instance is shared across multiple requests, it must be implemented in a thread-safe manner.

Member Summary

Constructors

```
MessageResources()
```

Methods

```
abstract Message getMessage(FacesContext context, java.lang.String messageId)  
    Return a localized Message0 instance corresponding to the specified parameters, or  
    null if the specified message identifier is not supported by the  
    MessageResources0 instance.  
  
abstract Message getMessage(FacesContext context, java.lang.String messageId,  
    java.lang.Object param0)  
    Return a localized Message0 instance corresponding to the specified parameters, or  
    null if the specified message identifier is not supported by the  
    MessageResources0 instance.  
  
abstract Message getMessage(FacesContext context, java.lang.String messageId,  
    java.lang.Object params)  
    Return a localized Message0 instance corresponding to the specified parameters, or  
    null if the specified message identifier is not supported by the  
    MessageResources0 instance.
```

Member Summary

```
abstract Message getMessage(FacesContext context, java.lang.String messageId,
                           java.lang.Object param0, java.lang.Object param1)
                           Return a localized Message0 instance corresponding to the specified parameters, or
                           null if the specified message identifier is not supported by the
                           MessageResources0 instance.

abstract Message getMessage(FacesContext context, java.lang.String messageId,
                           java.lang.Object param0, java.lang.Object param1,
                           java.lang.Object param2)
                           Return a localized Message0 instance corresponding to the specified parameters, or
                           null if the specified message identifier is not supported by the
                           MessageResources0 instance.

abstract Message getMessage(FacesContext context, java.lang.String messageId,
                           java.lang.Object param0, java.lang.Object param1,
                           java.lang.Object param2, java.lang.Object param3)
                           Return a localized Message0 instance corresponding to the specified parameters, or
                           null if the specified message identifier is not supported by the
                           MessageResources0 instance.
```

Inherited Member Summary

Methods inherited from class Object

```
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(),
toString(), wait(), wait(), wait()
```

Constructors

MessageResources()

```
public MessageResources()
```

Methods

getMessage(FacesContext, String)

```
public abstract javax.faces.context.Message0
                     getMessage(javax.faces.context.FacesContext0 context,
                               java.lang.String messageId)
```

Return a localized Message₀ instance corresponding to the specified parameters, or null if the specified message identifier is not supported by the MessageResources₀ instance.

Parameters:

context - The FacesContext₀ associated with the request being processed
messageId - Message identifier of the requested message

Throws:

NullPointerException - if context or messageId is null

getMessage(FacesContext, String, Object)

```
public abstract javax.faces.context.Message0
    getMessage(javax.faces.context.FacesContext0 context,
               java.lang.String messageId, java.lang.Object param0)
```

Return a localized Message₀ instance corresponding to the specified parameters, or null if the specified message identifier is not supported by the MessageResources₀ instance.

Parameters:

context - The FacesContext₀ associated with the request being processed
messageId - Message identifier of the requested message
param0 - First ubstitution parameter for this message

Throws:

NullPointerException - if context or messageId is null

getMessage(FacesContext, String, Object[])

```
public abstract javax.faces.context.Message0
    getMessage(javax.faces.context.FacesContext0 context,
               java.lang.String messageId, java.lang.Object[] params)
```

Return a localized Message₀ instance corresponding to the specified parameters, or null if the specified message identifier is not supported by the MessageResources₀ instance.

Parameters:

context - The FacesContext₀ associated with the request being processed
messageId - Message identifier of the requested message
params - Substitution parameters for this message

Throws:

NullPointerException - if context or messageId is null

getMessage(FacesContext, String, Object, Object)

```
public abstract javax.faces.context.Message0
    getMessage(javax.faces.context.FacesContext0 context,
```

```
java.lang.String messageId, java.lang.Object param0,
java.lang.Object param1)
```

Return a localized `Message` instance corresponding to the specified parameters, or null if the specified message identifier is not supported by the `MessageResources` instance.

Parameters:

- context - The `FacesContext` associated with the request being processed
- messageId - Message identifier of the requested message
- param0 - First substitution parameter for this message
- param1 - Second substitution parameter for this message

Throws:

- NullPointerException - if context or messageId is null

getMessage(FacesContext, String, Object, Object, Object)

```
public abstract javax.faces.context.Message,
    getMessage(javax.faces.context.FacesContext context,
        java.lang.String messageId, java.lang.Object param0,
        java.lang.Object param1, java.lang.Object param2)
```

Return a localized `Message` instance corresponding to the specified parameters, or null if the specified message identifier is not supported by the `MessageResources` instance.

Parameters:

- context - The `FacesContext` associated with the request being processed
- messageId - Message identifier of the requested message
- param0 - First substitution parameter for this message
- param1 - Second substitution parameter for this message
- param2 - Third substitution parameter for this message

Throws:

- NullPointerException - if context or messageId is null

getMessage(FacesContext, String, Object, Object, Object, Object)

```
public abstract javax.faces.context.Message,
    getMessage(javax.faces.context.FacesContext context,
        java.lang.String messageId, java.lang.Object param0,
        java.lang.Object param1, java.lang.Object param2,
        java.lang.Object param3)
```

Return a localized `Message` instance corresponding to the specified parameters, or null if the specified message identifier is not supported by the `MessageResources` instance.

Parameters:

context - The FacesContext₀ associated with the request being processed
messageId - Message identifier of the requested message
param0 - First ubstitution parameter for this message
param1 - Second substitution parameter for this message
param2 - Third substitution parameter for this message
param3 - Fourth substitution parameter for this message

Throws:

NullPointerException - if context or messageId is null

javax.faces.context MessageResourcesFactory

Declaration

```
public abstract class MessageResourcesFactory  
  
java.lang.Object  
|  
+-- javax.faces.context.MessageResourcesFactory
```

Description

MessageResourcesFactory is a Factory object that creates (if needed) and returns `MessageResources0` instances. Implementations of JavaServer Faces must provide `MessageResources0` instances for the following standard message resources identifiers:

- **MessageResourcesFactory.FACES_API_MESSAGES** - Messages whose message identifiers are defined in the JavaServer Faces specification, for messages generated by `javax.faces.*` concrete classes.
- **MessageResourcesFactory.FACES_IMPL_MESSAGES** - Messages whose message identifiers are defined by the JavaServer Faces implementation being utilized.

There must be one `MessageResourcesFactory0` instance per web application that is utilizing JavaServer Faces. This instance can be acquired, in a portable manner, by calling:

```
MessageResourcesFactory factory = (MessageResourcesFactory)  
    FactoryFinder.getFactory(FactoryFinder.MESSAGE_RESOURCES_FACTORY);
```

Member Summary

Fields

static FACES_API_MESSAGES ₀	
java.lang.String	Message resources identifier for a <code>MessageResources₀</code> instance containing messages whose message identifiers are defined in the JavaServer Faces specification.
static FACES_IMPL_MESSAGES ₀	
java.lang.String	Message resources identifier for a <code>MessageResources₀</code> instance containing messages whose message identifiers are defined by the JavaServer Faces implementation being used.

Constructors

```
MessageResourcesFactory( )0
```

Methods

Member Summary

abstract void addMessageResources(java.lang.String messageResourcesId, MessageResources messageResources) _o	Register a new MessageResources _o instance, associated with the specified messageResourcesId, to be supported by this MessageResourcesFactory _o .
abstract MessageResources getMessageResources(java.lang.String messageResourcesId) _o	Create (if needed) and return a MessageResources _o instance for the specified message resources identifier.
abstract java.util.Iterator getMessageResourcesIds() _o	Return an Iterator over the set of message resource identifiers supported by this factory.

Inherited Member Summary

Methods inherited from class Object

clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(),
toString(), wait(), wait(), wait()

Fields

FACES_API_MESSAGES

public static final java.lang.String FACES_API_MESSAGES

Message resources identifier for a MessageResources_o instance containing messages whose
message identifiers are defined in the JavaServer Faces specification.

FACES_IMPL_MESSAGES

public static final java.lang.String FACES_IMPL_MESSAGES

Message resources identifier for a MessageResources_o instance containing messages whose
message identifiers are defined by the JavaServer Faces implementation being used.

Constructors

MessageResourcesFactory()

public MessageResourcesFactory()

Methods

addMessageResources(String, MessageResources)

```
public abstract void addMessageResources(java.lang.String messageResourcesId,  
                                         javax.faces.context.MessageResources0 messageResources)
```

Register a new `MessageResources0` instance, associated with the specified `messageResourcesId`, to be supported by this `MessageResourcesFactory0`. This method may be called at any time, and makes the corresponding `MessageResources0` instance available throughout the remaining lifetime of this web application.

Parameters:

`messageResourcesId` - Identifier of the new `MessageResources0`
`messageResources` - `MessageResources0` instance that we are registering

Throws:

`java.lang.IllegalArgumentException` - if `messageResourcesId` is already registered in this `MessageResourcesFactory`
`NullPointerException` - if `messageResourcesId` or `messageResources` is null

getMessageResources(String)

```
public abstract javax.faces.context.MessageResources0  
                  getMessageResources(java.lang.String messageResourcesId)
```

Create (if needed) and return a `MessageResources0` instance for the specified message resources identifier. The set of available message resources identifiers is available via the `getMessageResourcesIds()` method.

Each call to `getMessageResources()` for the same `messageResourcesId`, from within the same web application, must return the same `MessageResources` instance.

Parameters:

`messageResourcesId` - Identifier of the requested `MessageResources0` instance

Throws:

`java.lang.IllegalArgumentException` - if no `MessageResources0` instance can be returned for the specified identifier
`NullPointerException` - if `messageResourcesId` is null

getMessageResourcesIds()

```
public abstract java.util.Iterator getMessageResourcesIds()
```

Return an `Iterator` over the set of message resource identifiers supported by this factory. This `Iterator` must include the standard identifiers described above.

javax.faces.context ResponseStream

Declaration

```
public abstract class ResponseStream extends java.io.OutputStream

java.lang.Object
|
+--java.io.OutputStream
|
+--javax.faces.context.ResponseStream
```

Description

ResponseStream is an interface describing an adapter to an underlying output mechanism for binary output.

Member Summary

Constructors

`ResponseStream()`

Inherited Member Summary

Methods inherited from class Object

`clone()`, `equals(Object)`, `finalize()`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`,
`toString()`, `wait()`, `wait()`, `wait()`

Methods inherited from class OutputStream

`close()`, `flush()`, `write(byte[], int, int)`, `write(byte[], int, int)`, `write(byte[], int, int)`

Constructors

ResponseStream()

```
public ResponseStream()
```

javax.faces.context ResponseWriter

Declaration

```
public abstract class ResponseWriter extends java.io.Writer  
  
java.lang.Object  
|  
+--java.io.Writer  
|  
+--javax.faces.context.ResponseWriter
```

Direct Known Subclasses: javax.faces.webapp.ResponseWriterBase

Description

ResponseWriter is an abstract class describing an adapter to an underlying output mechanism for character-based output. In addition to the low-level `write()` methods inherited from `java.io>Writer`, this class provides utility methods that are useful in producing elements and attributes for markup languages like HTML and XML.

Member Summary

Constructors

```
ResponseWriter()
```

Methods

```
abstract void endDocument()  
          Write whatever text should end a response.  
abstract void endElement(java.lang.String name)  
          Write the end of an element, after closing any open element created by a call to  
          startElement().  
abstract void startDocument()  
          Write whatever text should begin a response.  
abstract void startElement(java.lang.String name)  
          Write the start of an element, up to and including the element name.  
abstract void writeAttribute(java.lang.String name, java.lang.Object  
                           value)  
          Write an attribute name and corresponding value (after converting that text to a String  
          if necessary), after escaping it properly.  
abstract void writeComment(java.lang.Object comment)  
          Write a comment containing the specified text, after converting that text to a String if  
          necessary.
```

Member Summary

```
abstract void writeText(char text)_
           Write a single character, after escaping it properly.
abstract void writeText(char[] text)_
           Write text from a character array, after escaping it properly.
abstract void writeText(char[] text, int off, int len)_
           Write text from a character array, after escaping it properly for this method.
abstract void writeText(java.lang.Object text)_
           Write an object (after converting it to a String, if necessary), after escaping it properly.
abstract void writeURIAttribute(java.lang.String name, java.lang.Object
                                value)_
           Write a URI attribute name and corresponding value (after converting that text to a
           String if necessary), after encoding it properly (for example, '%' encoded for HTML).
```

Inherited Member Summary

Fields inherited from class `Writer`

`lock`

Methods inherited from class `Object`

`clone()`, `equals(Object)`, `finalize()`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`,
`toString()`, `wait()`, `wait()`, `wait()`

Methods inherited from class `Writer`

`close()`, `flush()`, `write(String, int, int)`, `write(String, int, int)`, `write(String,`
`int, int)`, `write(String, int, int)`, `write(String, int, int)`

Constructors

`ResponseWriter()`

```
public ResponseWriter()
```

Methods

`endDocument()`

```
public abstract void endDocument()
                  throws IOException
```

Write whatever text should end a response. If there is an open element that has been created by a call to `startElement()`, that element will be closed first.

Throws:

`java.io.IOException` - if an input/output error occurs

endElement(String)

```
public abstract void endElement(java.lang.String name)
    throws IOException
```

Write the end of an element, after closing any open element created by a call to `startElement()`.

Parameters:

`name` - Name of the element to be ended

Throws:

`java.io.IOException` - if an input/output error occurs

`NullPointerException` - if `name` is null

startDocument()

```
public abstract void startDocument()
    throws IOException
```

Write whatever text should begin a response.

Throws:

`java.io.IOException` - if an input/output error occurs

startElement(String)

```
public abstract void startElement(java.lang.String name)
    throws IOException
```

Write the start of an element, up to and including the element name. Once this method has been called, clients can call `writeAttribute()` or `writeURIAttribute()` method to add attributes and corresponding values. The starting element will be closed (that is, the trailing '`>`' character added) on any subsequent call to `startElement()`, `writeComment()`, `writeText()`, `endElement()`, or `endDocument()`.

Parameters:

`name` - Name of the element to be started

Throws:

`java.io.IOException` - if an input/output error occurs

`NullPointerException` - if `name` is null

writeAttribute(String, Object)

```
public abstract void writeAttribute(java.lang.String name, java.lang.Object value)
throws IOException
```

Write an attribute name and corresponding value (after converting that text to a String if necessary), after escaping it properly. This method may only be called after a call to `startElement()`, and before the opened element has been closed.

Parameters:

name - Attribute name to be added

value - Attribute value to be added

Throws:

`java.lang.IllegalStateException` - if this method is called when there is no currently open element

`java.io.IOException` - if an input/output error occurs

`NullPointerException` - if name or value is null

writeComment(Object)

```
public abstract void writeComment(java.lang.Object comment)
throws IOException
```

Write a comment containing the specified text, after converting that text to a String if necessary. If there is an open element that has been created by a call to `startElement()`, that element will be closed first.

Parameters:

comment - Text content of the comment

Throws:

`java.io.IOException` - if an input/output error occurs

`NullPointerException` - if comment is null

writeText(char)

```
public abstract void writeText(char text)
throws IOException
```

Write a single character, after escaping it properly. If there is an open element that has been created by a call to `startElement()`, that element will be closed first.

Parameters:

text - Text to be written

Throws:

`java.io.IOException` - if an input/output error occurs

writeText(char[])

```
public abstract void writeText(char[ ] text)
    throws IOException
```

Write text from a character array, after escaping it properly. This is equivalent to calling `writeText(c, 0, c.length)`. If there is an open element that has been created by a call to `startElement()`, that element will be closed first.

Parameters:

`text` - Text to be written

Throws:

`java.io.IOException` - if an input/output error occurs
`NullPointerException` - if `text` is null

writeText(char[], int, int)

```
public abstract void writeText(char[ ] text, int off, int len)
    throws IOException
```

Write text from a character array, after escaping it properly for this method. If there is an open element that has been created by a call to `startElement()`, that element will be closed first.

Parameters:

`text` - Text to be written

`off` - Starting offset (zero-relative)

`len` - Number of characters to be written

Throws:

`java.lang.IndexOutOfBoundsException` - if the calculated starting or ending position is outside the bounds of the character array
`java.io.IOException` - if an input/output error occurs
`NullPointerException` - if `text` is null

writeText(Object)

```
public abstract void writeText(java.lang.Object text)
    throws IOException
```

Write an object (after converting it to a String, if necessary), after escaping it properly. If there is an open element that has been created by a call to `startElement()`, that element will be closed first.

Parameters:

`text` - Text to be written

Throws:

`java.io.IOException` - if an input/output error occurs
`NullPointerException` - if `text` is null

writeURIAttribute(String, Object)

```
public abstract void writeURIAttribute(java.lang.String name,  
                                      java.lang.Object value)  
                                      throws IOException
```

Write a URI attribute name and corresponding value (after converting that text to a String if necessary), after encoding it properly (for example, '%' encoded for HTML). This method may only be called after a call to `startElement()`, and before the opened element has been closed.

Parameters:

`name` - Attribute name to be added
`value` - Attribute value to be added

Throws:

`java.lang.IllegalStateException` - if this method is called when there is no currently open element
`java.io.IOException` - if an input/output error occurs
`NullPointerException` - if `name` or `value` is null

JSF.10.4 Package javax.faces.convert

Package **javax.faces.convert**

Description

Classes and interfaces defining converters.

Class Summary

Interfaces

`Converter0`

Converter is an interface describing a Java class that can perform Object-to-String and String-to-Object conversions between model data objects and a String representation of those objects that is suitable for rendering.

Classes

`ConverterFactory0`

ConverterFactory is a factory object that creates (if needed) and returns new `Converter0` instances.

Exceptions

`ConverterException0`

ConverterException is an exception thrown by the `getAsObject()` or `getAsString()` method of a `Converter0`, to indicate that the requested conversion cannot be performed.

javax.faces.convert Converter

Declaration

```
public interface Converter extends java.io.Serializable
```

All Superinterfaces: `java.io.Serializable`

Description

Converter is an interface describing a Java class that can perform Object-to-String and String-to-Object conversions between model data objects and a String representation of those objects that is suitable for rendering. `Converter`s instances are shared, so they must be programmed in a thread-safe manner. In addition, because `Converter`s are part of the saved and restored state of a component tree, classes that implement this interface must also be serializable.

Member Summary

Methods

<pre>java.lang.Object getAsObject(javax.faces.context.FacesContext context, javax.faces.component.UIComponent component, java.lang.String value)</pre>	Convert the specified string value, which is associated with the specified <code>javax.faces.component.UIComponent</code> , into a model data object that is appropriate for being stored during the <i>Apply Request Values</i> phase of the request processing lifecycle.
<pre>java.lang.String getAsString(javax.faces.context.FacesContext context, javax.faces.component.UIComponent component, java.lang.Object value)</pre>	Convert the specified model object value, which is associated with the specified <code>javax.faces.component.UIComponent</code> , into a String that is suitable for being included in the response generated during the <i>Render Response</i> phase of the request processing lifecycle.

Methods

getAsObject(FacesContext, UIComponent, String)

```
public java.lang.Object getAsObject(javax.faces.context.FacesContext0 context,  
                                javax.faces.component.UIComponent0 component, java.lang.String value)  
                                throws ConverterException
```

Convert the specified string value, which is associated with the specified `javax.faces.component.UIComponent0`, into a model data object that is appropriate for being stored during the *Apply Request Values* phase of the request processing lifecycle.

Parameters:

context - `javax.faces.context.FacesContext0` for the request being processed
component - `javax.faces.component.UIComponent0` with which this model object value is associated
value - String value to be converted (may be null)

Throws:

`ConverterException0` - if conversion cannot be successfully performed
`NullPointerException` - if context or component is null

AsString(FacesContext, UIComponent, Object)

```
public java.lang.String AsString(javax.faces.context.FacesContext0 context,  
                                javax.faces.component.UIComponent0 component, java.lang.Object value)  
                                throws ConverterException
```

Convert the specified model object value, which is associated with the specified `javax.faces.component.UIComponent0`, into a String that is suitable for being included in the response generated during the *Render Response* phase of the request processing lifecycle.

Parameters:

context - `javax.faces.context.FacesContext0` for the request being processed
component - `javax.faces.component.UIComponent0` with which this model object value is associated
value - Model object value to be converted (may be null)

Throws:

`ConverterException0` - if conversion cannot be successfully performed
`NullPointerException` - if context or component is null

javax.faces.convert ConverterException

Declaration

```
public class ConverterException extends javax.faces.FacesException,  
  
java.lang.Object  
|  
+--java.lang.Throwable  
|  
+--java.lang.Exception  
|  
+--java.lang.RuntimeException  
|  
+--javax.faces.FacesException,  
|  
+--javax.faces.convert.ConverterException
```

All Implemented Interfaces: java.io.Serializable

Description

ConverterException is an exception thrown by the `getAsObject()` or `getAsString()` method of a `Converter`, to indicate that the requested conversion cannot be performed.

Member Summary

Constructors

<code>ConverterException()</code>	Construct a new exception with no detail message or root cause.
<code>ConverterException(java.lang.String message)</code>	Construct a new exception with the specified detail message and no root cause.
<code>ConverterException(java.lang.String message, java.lang.Throwable cause)</code>	Construct a new exception with the specified detail message and root cause.
<code>ConverterException(java.lang.Throwable cause)</code>	Construct a new exception with the specified root cause.

Inherited Member Summary

Methods inherited from interface `FacesException`

`getCause()`

Methods inherited from class `Object`

`clone()`, `equals(Object)`, `finalize()`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`

Methods inherited from class `Throwable`

`fillInStackTrace()`, `getLocalizedMessage()`, `getMessage()`,
`printStackTrace(PrintWriter)`, `printStackTrace(PrintWriter)`,
`printStackTrace(PrintWriter)`, `toString()`

Constructors

`ConverterException()`

`public ConverterException()`

Construct a new exception with no detail message or root cause.

`ConverterException(String)`

`public ConverterException(java.lang.String message)`

Construct a new exception with the specified detail message and no root cause.

Parameters:

`message` - The detail message for this exception

`ConverterException(String, Throwable)`

`public ConverterException(java.lang.String message, java.lang.Throwable cause)`

Construct a new exception with the specified detail message and root cause.

Parameters:

`message` - The detail message for this exception

`cause` - The root cause for this exception

`ConverterException(Throwable)`

`public ConverterException(java.lang.Throwable cause)`

Construct a new exception with the specified root cause. The detail message will be set to (cause == null ? null : cause.toString())

Parameters:

cause - The root cause for this exception

javax.faces.convert ConverterFactory

Declaration

```
public abstract class ConverterFactory  
  
java.lang.Object  
|  
+--javax.faces.convertConverterFactory
```

Description

ConverterFactory is a factory object that creates (if neeeded) and returns new *Converter*₀ instances.

There must be one *ConverterFactory* instance per web application that is utilizing JavaServer Faces. This instance can be acquired, in a portable manner, by calling:

```
ConverterFactory factory = (ConverterFactory)  
    FactoryFinder.getFactory(FactoryFinder.CONVERTER_FACTORY);
```

Member Summary

Constructors

```
ConverterFactory()
```

Methods

```
abstract void addConverter(java.lang.Class clazz, Converter converter)0  
Register a new Converter0 instance that knows how to perform conversions to and  
from the specified object class.
```

```
abstract void addConverter(java.lang.String converterId, Converter  
    converter)0  
Register a new Converter0 instance with the specified converterId, to be  
supported by this ConverterFactory.
```

```
abstract Converter getConverter(java.lang.Class clazz)0  
Create (if needed) and return a Converter0 instance that may be used to perform  
Object-to-String and String-to-Object conversions for objects of the specified class.
```

```
abstract Converter getConverter(java.lang.String converterId)0  
Create (if needed) and return a Converter0 instance that may be used to perform  
Object-to-String and String-to-Object conversions for web applications based on  
JavaServer Faces.
```

```
abstract getConverterClasses()0  
java.util.Iterator  
Return an Iterator over the set of classes for which we have registered  
Converter0s in this factory.
```

Member Summary	
<code>abstract java.util.Iterator getConverterIds()</code>	Return an Iterator over the set of converter identifiers supported by this factory.

Inherited Member Summary	
Methods inherited from class Object	
<code>clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()</code>	

Constructors

ConverterFactory()

```
public ConverterFactory()
```

Methods

addConverter(Class, Converter)

```
public abstract void addConverter(java.lang.Class clazz,  
                                  javax.faces.convert.Converter<?> converter)
```

Register a new `Converter<?>` instance that knows how to perform conversions to and from the specified object class. This method may be called at any time, and makes the corresponding `Converter<?>` instance available throughout the remaining lifetime of this web application.

Parameters:

`clazz` - Class for which Object-to-String and String-to-Object conversions are supported by this `Converter<?>`
`converter` - `Converter<?>` instance that we are registering

Throws:

`NullPointerException` - if `clazz` or `converter` is null

addConverter(String, Converter)

```
public abstract void addConverter(java.lang.String converterId,  
                                  javax.faces.convert.Converter<?> converter)
```

Register a new `Converter` instance with the specified `converterId`, to be supported by this `ConverterFactory`. This method may be called at any time, and makes the corresponding `Converter` instance available throughout the remaining lifetime of this web application.

Parameters:

`converterId` - Identifier of the new `Converter`
 `converter` - `Converter` instance that we are registering

Throws:

`java.lang.IllegalArgumentException` - if a `Converter` with the specified `converterId` has already been registered

 `NullPointerException` - if `converterId` or `converter` is null

getConverter(Class)

```
public abstract javax.faces.convert.Converter getConverter(java.lang.Class clazz)
```

Create (if needed) and return a `Converter` instance that may be used to perform Object-to-String and String-to-Object conversions for objects of the specified class. A search of the registered Converters is performed in the following order:

- An exact match on the original class.
- An exact match on one of the interfaces directly implemented by the original class, checked in the order they are returned by `Class.getInterfaces()`.
- Recursively for each superclass of the original class:
 - An exact match on this superclass.
 - An exact match on one of the interfaces directly implemented by this superclass.

If a `Converter` is discovered during the search described above, it is returned; otherwise null is returned.

Each call to `getConverter()` for the same `clazz`, from within the same web application, must return the same `Converter` instance.

Parameters:

`clazz` - Class for which a `Converter` is requested

Throws:

`NullPointerException` - if `clazz` is null

getConverter(String)

```
public abstract javax.faces.convert.Converter getConverter(java.lang.String converterId)
```

Create (if needed) and return a `Converter0` instance that may be used to perform Object-to-String and String-to-Object conversions for web applications based on JavaServer Faces. The set of available converter identifiers is available via the `getConverterIds()` method. If no `Converter0` has been registered for the specified converter identifier, return null.

Each call to `getConverter()` for the same `converterId`, from within the same web application, must return the same `Converter0` instance.

Parameters:

`converterId` - Converter identifier of the requested `Converter0` instance

Throws:

`NullPointerException` - if `converterId` is null

getConverterClasses()

```
public abstract java.util.Iterator getConverterClasses()
```

Return an `Iterator` over the set of classes for which we have registered `Converter0`s in this factory.

getConverterIds()

```
public abstract java.util.Iterator getConverterIds()
```

Return an `Iterator` over the set of converter identifiers supported by this factory.

JSF.10.5 Package `javax.faces.event`

Package `javax.faces.event`

Description

Interfaces describing events and event listeners, and concrete event implementation classes.

Class Summary

Interfaces

<code>ActionListener₀</code>	A listener interface for receiving <code>ActionEvent₀</code> s.
<code>FacesListener₀</code>	A generic base interface for event listeners for various types of <code>FacesEvent₀</code> s.
<code>ValueChangeListener₀</code>	A listener interface for receiving <code>ValueChangeEvent₀</code> s.

Classes

<code>ActionEvent₀</code>	An <code>ActionEvent₀</code> represents the activation of a user interface component (such as a <code>UICommand</code>).
<code>ApplicationEvent₀</code>	ApplicationEvent is a subclass of <code>FacesEvent₀</code> that describes an event queued in any request processing lifecycle phase up to (and including) the <i>Invoke Application</i> phase, and is handled during the <i>Invoke Application</i> phase.
<code>CommandEvent₀</code>	CommandEvent is a subclass of <code>ApplicationEvent₀</code> that indicates that a particular <code>javax.faces.component.UICommand₀</code> was selected by the user.
<code>FacesEvent₀</code>	FacesEvent is the base class for user interface and application events that can be fired by <code>javax.faces.component.UIComponent₀</code> s.
<code>FormEvent₀</code>	FormEvent is a subclass of <code>ApplicationEvent₀</code> that indicates that a particular <code>javax.faces.component.UIForm₀</code> was submitted by the user.
<code>PhaseId₀</code>	Typesafe enumeration of the legal values that may be returned by the <code>getPhaseId()</code> method of the <code>FacesListener₀</code> interface.
<code>ValueChangeEvent₀</code>	A <code>ValueChangeEvent₀</code> is a notification that the local value of the source component has been changed as a result of user interface activity.

Exceptions

<code>AbortProcessingException₀</code>	An exception that may be thrown by event listeners to terminate the processing of the current event.
---	--

`javax.faces.event`

AbortProcessingException

Declaration

```
public class AbortProcessingException extends javax.faces.FacesException0
java.lang.Object
|
+--java.lang.Throwable
|
+--java.lang.Exception
|
+--java.lang.RuntimeException
|
+--javax.faces.FacesException0
|
+--javax.faces.event.AbortProcessingException
```

All Implemented Interfaces: `java.io.Serializable`

Description

An exception that may be thrown by event listeners to terminate the processing of the current event.

Member Summary

Constructors

`AbortProcessingException()`

Inherited Member Summary

Methods inherited from interface `FacesException0`

`getCause()`

Methods inherited from class `Object`

`clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait()`

Methods inherited from class `Throwable`

Inherited Member Summary

```
fillInStackTrace(), getLocalizedMessage(), getMessage(),
printStackTrace(PrintWriter), printStackTrace(PrintWriter),
printStackTrace(PrintWriter), toString()
```

Constructors

AbortProcessingException()

```
public AbortProcessingException()
```

javax.faces.event ActionEvent

Declaration

```
public class ActionEvent extends FacesEvent,  
  
java.lang.Object  
|  
+--java.util.EventObject  
|  
+--javax.faces.event.FacesEvent,  
|  
+--javax.faces.event.ActionEvent
```

All Implemented Interfaces: java.io.Serializable

Description

An ActionEvent₀ represents the activation of a user interface component (such as a UICommand).

Member Summary

Constructors

```
ActionEvent(javax.faces.component.UIComponent source,  
java.lang.String actionCommand),  
Construct a new event object from the specified source component and action  
command.
```

Methods

```
java.lang.String getActionCommand(),  
Return the String identifying the command that was activated.
```

Inherited Member Summary

Fields inherited from class EventObject

source

Methods inherited from class EventObject

getSource(), toString()

Inherited Member Summary

Methods inherited from class `FacesEvent`

`getComponent()`

Methods inherited from class `Object`

`clone()`, `equals(Object)`, `finalize()`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`

Constructors

`ActionEvent(UIComponent, String)`

```
public ActionEvent(javax.faces.component.UIComponent source,  
                   java.lang.String actionCommand)
```

Construct a new event object from the specified source component and action command.

Parameters:

`component` - Source `javax.faces.component.UIComponent` for this event
`actionCommand` - String identifying the command that was activated

Throws:

`java.lang.IllegalArgumentException` - if `component` is null

Methods

`getActionCommand()`

```
public java.lang.String getActionCommand()
```

Return the String identifying the command that was activated.

javax.faces.event ActionListener

Declaration

```
public interface ActionListener extends FacesListener,
```

All Superinterfaces: `java.util.EventListener, FacesListener,`
`java.io.Serializable`

Description

A listener interface for receiving `ActionEvent`s. A class that is interested in receiving such events implements this interface, and then registers itself with the source

`javax.faces.component.UIComponent` of interest, by calling `addActionListener()`.

Member Summary

Methods

```
void processAction(ActionEvent event)
    Invoked when the action described by the specified ActionEvent occurs.
```

Inherited Member Summary

Methods inherited from interface `FacesListener`

```
getPhaseId()
```

Methods

processAction(ActionEvent)

```
public void processAction(javax.faces.event.ActionEvent event)
    throws AbortProcessingException
```

Invoked when the action described by the specified `ActionEvent` occurs.

Parameters:

`event` - The `ActionEvent` that has occurred

Throws:

`AbortProcessingException` - Signal the JavaServer Faces implementation that no further processing on the current event should be performed

javax.faces.event ApplicationEvent

Declaration

```
public class ApplicationEvent extends FacesEvent,  
  
java.lang.Object  
|  
+--java.util.EventObject  
|  
+--javax.faces.event.FacesEvent,  
|  
+--javax.faces.event.ApplicationEvent
```

All Implemented Interfaces: java.io.Serializable

Direct Known Subclasses: CommandEvent,

Description

Deprecated. The current mechanism for handling application events is a placeholder, and will be replaced in the next public release of JavaServer Faces.

ApplicationEvent is a subclass of `FacesEvent`, that describes an event queued in any request processing lifecycle phase up to (and including) the *Invoke Application* phase, and is handled during the *Invoke Application* phase.

Member Summary

Constructors

```
ApplicationEvent(javax.faces.component.UIComponent source),  
Construct a new event object from the specified source component.
```

Methods

```
java.lang.String toString(),  
Return a String rendition of this object value.
```

Inherited Member Summary

Fields inherited from class `EventObject`

`source`

Methods inherited from class `EventObject`

`getSource()`

Methods inherited from class `FacesEvent`

`getComponent()`

Methods inherited from class `Object`

`clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait()`

Constructors

`ApplicationEvent(UIComponent)`

```
public ApplicationEvent(javax.faces.component.UIComponent0 source)
```

Construct a new event object from the specified source component.

Parameters:

`component` - Source `javax.faces.component.UIComponent0` for this event

Throws:

`java.lang.IllegalArgumentException` - if component is null

Methods

`toString()`

```
public java.lang.String toString()
```

Return a String rendition of this object value.

Overrides: `toString` in class `EventObject`

javax.faces.event CommandEvent

Declaration

```
public class CommandEvent extends ApplicationEvent,  
  
java.lang.Object  
|  
+--java.util.EventObject  
|  
+--javax.faces.event.FacesEvent,  
|  
+--javax.faces.event.ApplicationEvent,  
|  
+--javax.faces.event.CommandEvent
```

All Implemented Interfaces: java.io.Serializable

Direct Known Subclasses: FormEvent,

Description

Deprecated. The current mechanism for handling application events is a placeholder, and will be replaced in the next public release of JavaServer Faces.

CommandEvent is a subclass of `ApplicationEvent`, that indicates that a particular `javax.faces.component.UICommand` was selected by the user. It is queued to the application, for processing during the *Invoke Application* phase of the request processing lifecycle.

Member Summary	
Fields	
protected java.lang.String	commandName, The command name whose selection this event signifies.
Constructors	
	CommandEvent(javax.faces.component.UIComponent source, java.lang.String commandName), Construct a new event object from the specified source component.
Methods	

Member Summary

```
java.lang.String getCommandName()
```

Return the command name of the javax.faces.component.UICommand whose selection this event signifies.

```
java.lang.String toString()
```

Inherited Member Summary**Fields inherited from class EventObject**

source

Methods inherited from class EventObject

```
getSource()
```

Methods inherited from class FacesEvent

```
getComponent()
```

Methods inherited from class Object

```
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(),  
wait(), wait(), wait()
```

Fields

commandName

```
protected java.lang.String commandName
```

The command name whose selection this event signifies.

Constructors

CommandEvent(UIComponent, String)

```
public CommandEvent(javax.faces.component.UIComponent source,  
                    java.lang.String commandName)
```

Construct a new event object from the specified source component.

Parameters:

component - Source javax.faces.component.UIComponent for this event

`commandName` - Command name of the command this event signifies

Throws:

`java.lang.IllegalArgumentException` - if component is null

`NullPointerException` - if `commandName` is null

Methods

getCommandName()

```
public java.lang.String getCommandName()
```

Return the command name of the `javax.faces.component.UICommand`, whose selection this event signifies.

toString()

```
public java.lang.String toString()
```

Description copied from class: `javax.faces.event.ApplicationEvent`

Return a String rendition of this object value.

Overrides: `toString()` in class `ApplicationEvent`

javax.faces.event FacesEvent

Declaration

```
public class FacesEvent extends java.util.EventObject  
  
java.lang.Object  
|  
+--java.util.EventObject  
|  
+--javax.faces.event.FacesEvent
```

All Implemented Interfaces: java.io.Serializable

Direct Known Subclasses: ActionEvent₀, ApplicationEvent₀, ValueChangedEvent₀

Description

FacesEvent is the base class for user interface and application events that can be fired by javax.faces.component.UIComponent₀s. Concrete event classes must subclass FacesEvent₀ in order to be supported by the request processing lifecycle.

Member Summary

Constructors

```
FacesEvent( javax.faces.component.UIComponent source )0  
Construct a new event object from the specified source component.
```

Methods

javax.faces.component. UIComponent	<code>getComponent()₀</code> Return the source javax.faces.component.UIComponent ₀ that sent this event.
---------------------------------------	---

Inherited Member Summary

Fields inherited from class EventObject

source

Inherited Member Summary

Methods inherited from class `EventObject`

`getSource()`, `toString()`

Methods inherited from class `Object`

`clone()`, `equals(Object)`, `finalize()`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`

Constructors

`FacesEvent(UIComponent)`

```
public FacesEvent(javax.faces.component.UIComponent0 source)
```

Construct a new event object from the specified source component.

Parameters:

`component` - Source `javax.faces.component.UIComponent0` for this event

Throws:

`java.lang.IllegalArgumentException` - if `component` is null

Methods

`getComponent()`

```
public javax.faces.component.UIComponent0 getComponent()
```

Return the source `javax.faces.component.UIComponent0` that sent this event.

javax.faces.event FacesListener

Declaration

```
public interface FacesListener extends java.util.EventListener, java.io.Serializable
```

All Superinterfaces: java.util.EventListener, java.io.Serializable

All Known Subinterfaces: ActionListener₀, ValueChangedListener₀

Description

A generic base interface for event listeners for various types of FacesEvent₀s. All listener interfaces for specific FacesEvent₀ event types must extend this interface.

Member Summary

Methods

PhaseId getPhaseId()

Return the identifier of the request processing phase during which this listener is interested in processing events of the type we are supporting.

Methods

getPhaseId()

```
public javax.faces.event.PhaseId0 getPhaseId()
```

Return the identifier of the request processing phase during which this listener is interested in processing events of the type we are supporting. Legal values are the singleton instances defined by the PhaseId₀ class.

FacesListener₀ instances should generally return PhaseId.ANY_PHASE to be notified about events no matter what the current request processing phase is.

javax.faces.event FormEvent

Declaration

```
public class FormEvent extends CommandEvent0
    java.lang.Object
        |
        +--java.util.EventObject
            |
            +--javax.faces.event.FacesEvent0
                |
                +--javax.faces.event.ApplicationEvent0
                    |
                    +--javax.faces.event.CommandEvent0
                        |
                        +--javax.faces.event.FormEvent
```

All Implemented Interfaces: java.io.Serializable

Description

Deprecated. The current mechanism for handling application events is a placeholder, and will be replaced in the next public release of JavaServer Faces.

FormEvent is a subclass of ApplicationEvent₀ that indicates that a particular javax.faces.component.UIForm₀ was submitted by the user. It is queued to the application, for processing during the *Invoke Application* phase of the request processing lifecycle.

Member Summary

Constructors

```
FormEvent( javax.faces.component.UIComponent source,
           java.lang.String formName, java.lang.String commandName )0
           Construct a new event object from the specified source component.
```

Methods

```
java.lang.String getFormName()0
           Return the form name of the javax.faces.component.UIForm0 whose
           submission this event signifies.

java.lang.String toString()0
           Return a String rendition of this object value.
```

Inherited Member Summary

Fields inherited from class `CommandEvent`

`commandName`

Fields inherited from class `EventObject`

`source`

Methods inherited from class `CommandEvent`

`getCommandName()`

Methods inherited from class `EventObject`

`getSource()`

Methods inherited from class `FacesEvent`

`getComponent()`

Methods inherited from class `Object`

`clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), wait(), wait(), wait()`

Constructors

`FormEvent(UIComponent, String, String)`

```
public FormEvent(javax.faces.component.UIComponent source,  
                 java.lang.String formName, java.lang.String commandName)
```

Construct a new event object from the specified source component.

Parameters:

`component` - Source `javax.faces.component.UIComponent` for this event

`formName` - Form name of the form this event signifies

`commandName` - Command name of the submit button that caused this form to be submitted

Throws:

`java.lang.IllegalArgumentException` - if `component` is null

`NullPointerException` - if `commandName` or `formName` is null

Methods

getFormName()

```
public java.lang.String getFormName()
```

Return the form name of the `javax.faces.component.UIForm` whose submission this event signifies.

toString()

```
public java.lang.String toString()
```

Return a String rendition of this object value.

Overrides: `toString()` in class `CommandEvent`

javax.faces.event PhaseId

Declaration

```
public class PhaseId implements java.lang.Comparable  
  
java.lang.Object  
|  
+--javax.faces.event.PhaseId
```

All Implemented Interfaces: java.lang.Comparable

Description

Typesafe enumeration of the legal values that may be returned by the `getPhaseId()` method of the `FacesListener0` interface.

Member Summary

Fields

```
static PhaseId ANY_PHASE0  
    Identifier that indicates an interest in events, on matter which request processing phase  
    is being performed.  
static PhaseId APPLY_REQUEST_VALUES0  
    Identifier that indicates an interest in events queued during the Apply Request Values  
    phase of the request processing lifecycle.  
static PhaseId PROCESS_VALIDATIONS0  
    Identifier that indicates an interest in events queued during the Process Validations  
    phase of the request processing lifecycle.  
static PhaseId RECONSTITUTE_REQUEST0  
    Identifier that indicates an interest in events queued during the Reconstitute Request  
    phase of the request processing lifecycle.  
static PhaseId UPDATE_MODEL_VALUES0  
    Identifier that indicates an interest in events queued during the Update Model Values  
    phase of the request processing lifecycle.  
static java.util.List VALUES0  
    List of valid PhaseId0 instances, in ascending order of their ordinal value.
```

Methods

```
int compareTo(java.lang.Object other)  
    Compare this PhaseId0 instance to the specified one.  
int getOrdinal()0  
    Return the ordinal value of this PhaseId0 instance.
```

Member Summary

```
java.lang.String toString()
```

Return a String representation of this PhaseId₀ instance.

Inherited Member Summary**Methods inherited from class Object**

```
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(),
wait(), wait(), wait()
```

Fields

ANY_PHASE

```
public static final javax.faces.event.PhaseId0 ANY_PHASE
```

Identifier that indicates an interest in events, on matter which request processing phase is being performed.

APPLY_REQUEST_VALUES

```
public static final javax.faces.event.PhaseId0 APPLY_REQUEST_VALUES
```

Identifier that indicates an interest in events queued during the *Apply Request Values* phase of the request processing lifecycle.

PROCESS_VALIDATIONS

```
public static final javax.faces.event.PhaseId0 PROCESS_VALIDATIONS
```

Identifier that indicates an interest in events queued during the *Process Validations* phase of the request processing lifecycle.

RECONSTITUTE_REQUEST

```
public static final javax.faces.event.PhaseId0 RECONSTITUTE_REQUEST
```

Identifier that indicates an interest in events queued during the *Reconstitute Request* phase of the request processing lifecycle.

UPDATE_MODEL_VALUES

```
public static final javax.faces.event.PhaseId0 UPDATE_MODEL_VALUES
```

Identifier that indicates an interest in events queued during the *Update Model Values* phase of the request processing lifecycle.

VALUES

```
public static final java.util.List VALUES
```

List of valid `PhaseId0` instances, in ascending order of their ordinal value.

Methods

compareTo(Object)

```
public int compareTo(java.lang.Object other)
```

Compare this `PhaseId0` instance to the specified one. Returns a negative integer, zero, or a positive integer if this object is less than, equal to, or greater than the specified object.

Specified By: `compareTo` in interface `Comparable`

Parameters:

`other` - The other object to be compared to

getOrdinal()

```
public int getOrdinal()
```

Return the ordinal value of this `PhaseId0` instance.

toString()

```
public java.lang.String toString()
```

Return a String representation of this `PhaseId0` instance.

Overrides: `toString` in class `Object`

javax.faces.event ValueChangedEvent

Declaration

```
public class ValueChangedEvent extends FacesEvent,  
  
java.lang.Object  
|  
+--java.util.EventObject  
|  
+--javax.faces.event.FacesEvent,  
|  
+--javax.faces.event.ValueChangedEvent
```

All Implemented Interfaces: java.io.Serializable

Description

A ValueChangedEvent_o is a notification that the local value of the source component has been changed as a result of user interface activity. It is not fired unless validation of the new value was completed successfully.

Member Summary

Constructors

```
ValueChangedEvent(javax.faces.component.UIComponent  
component, java.lang.Object oldValue, java.lang.Object  
newValue),
```

Construct a new event object from the specified source component, old value, and new value.

Methods

```
java.lang.Object getNewValue(),
```

Return the current local value of the source

```
javax.faces.component.UIComponent,
```

```
java.lang.Object getOldValue(),
```

Return the previous local value of the source

```
javax.faces.component.UIComponent.
```

Inherited Member Summary

Fields inherited from class `EventObject`

source

Methods inherited from class `EventObject`

`getSource()`, `toString()`

Methods inherited from class `FacesEvent`

`getComponent()`

Methods inherited from class `Object`

`clone()`, `equals(Object)`, `finalize()`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `wait()`, `wait()`, `wait()`

Constructors

`ValueChangedEvent(UIComponent, Object, Object)`

```
public ValueChangedEvent(javax.faces.component.UIComponent, component,  
                        java.lang.Object oldValue, java.lang.Object newValue)
```

Construct a new event object from the specified source component, old value, and new value.

Parameters:

component - Source `javax.faces.component.UIComponent` for this event

oldValue - The previous local value of this

`javax.faces.component.UIComponent`,

newValue - The new local value of the `javax.faces.component.UIComponent`

Throws:

`java.lang.IllegalArgumentException` - if component is null

Methods

`getNewValue()`

```
public java.lang.Object getNewValue()
```

Return the current local value of the source `javax.faces.component.UIComponent`.

getOldValue()

```
public java.lang.Object getOldValue()
```

Return the previous local value of the source `javax.faces.component.UIComponent`.

javax.faces.event

ValueChangedListener

Declaration

```
public interface ValueChangeListener extends FacesListener,
```

All Superinterfaces: `java.util.EventListener`, `FacesListener`,
`java.io.Serializable`

Description

A listener interface for receiving `ValueChangeEvent`s. A class that is interested in receiving such events implements this interface, and then registers itself with the source `javax.faces.component.UIComponent` of interest, by calling `addValueChangeListener()`.

Member Summary

Methods

```
void processValueChanged(ValueChangeEvent event)
```

Invoked when the value change described by the specified `ValueChangeEvent` occurs.

Inherited Member Summary

Methods inherited from interface `FacesListener`

```
getPhaseId()
```

Methods

`processValueChanged(ValueChangeEvent)`

```
public void processValueChanged(javax.faces.event.ValueChangeEvent event)
                                throws AbortProcessingException
```

Invoked when the value change described by the specified `ValueChangeEvent` occurs.

Parameters:

event - The ValueChangedEvent₀ that has occurred

Throws:

AbortProcessingException₀ - Signal the JavaServer Faces implementation that no further processing on the current event should be performed

JSF.10.6 Package `javax.faces.lifecycle`

Package `javax.faces.lifecycle`

Description

Classes and interfaces defining lifecycle management for the JavaServer Faces implementation.

Class Summary

Interfaces

`ApplicationHandler0`

ApplicationHandler is an interface defining application callbacks that will be invoked during the *Invoke Application* phase of the request processing lifecycle.

`ViewHandler0`

ViewHandler is an interface defining a mechanism by which the *Render Response* phase of the request processing lifecycle can use a pluggable mechanism to support different response generation technologies.

Classes

`Lifecycle0`

Lifecycle is a state machine that manages the processing of the entire lifecycle of a particular JavaServer Faces request.

`LifecycleFactory0`

LifecycleFactory is a factory object that creates (if needed) and returns `Lifecycle0` instances.

`Phase0`

A **Phase** is a single step in the processing of a JavaServer Faces request throughout its entire `Lifecycle0`.

javax.faces.lifecycle ApplicationHandler

Declaration

```
public interface ApplicationHandler
```

Description

Deprecated. The current mechanism for handling application events is a placeholder, and will be replaced in the next public release of JavaServer Faces.

ApplicationHandler is an interface defining application callbacks that will be invoked during the *Invoke Application* phase of the request processing lifecycle. Each application event that has been queued to the `javax.faces.context.FacesContext` will trigger a call to one of the methods defined on this interface.

It is the responsibility of any application wishing to respond to application level events to register an instance of this interface, by calling the `setApplicationHandler()` method of the `Lifecycle` instance used to process incoming requests, prior to receiving the first request.

Member Summary

Methods

```
boolean processEvent(javax.faces.context.FacesContext context,
                     javax.faces.event.FacesEvent event)
```

Process a command event that has been queued for the application.

Methods

processEvent(FacesContext, FacesEvent)

```
public boolean processEvent(javax.faces.context.FacesContext context,
                           javax.faces.event.FacesEvent event)
```

Process a command event that has been queued for the application. Return `true` if control should be transferred directly to the *Render Response* phase (bypassing any remaining application events that might have been queued), or `false` to process any remaining events normally.

Parameters:

`context` - `FacesContext` for the current request

`event` - FacesEvent to be processed

Throws:

NullPointerException - if any parameter is null

javax.faces.lifecycle

Lifecycle

Declaration

```
public abstract class Lifecycle  
  
java.lang.Object  
|  
+--javax.faces.lifecycle.Lifecycle
```

Description

Lifecycle is a state machine that manages the processing of the entire lifecycle of a particular JavaServer Faces request. It is responsible for executing all of the phases that have been defined by the JavaServer Faces Specification, in the specified order, unless otherwise directed by activities that occurred during the execution of each phase.

An instance of `Lifecycle` is created by calling the `getLifecycle()` method of `LifecycleFactory0`, for a specified lifecycle identifier. Because this instance is shared across multiple simultaneous requests, it must be implemented in a thread-safe manner.

Member Summary

Constructors

```
Lifecycle()0
```

Methods

<pre>abstract void execute(javax.faces.context.FacesContext context)₀ abstract int executePhase(javax.faces.context.FacesContext context, Phase phase)₀ abstract ApplicationHandler getApplicationHandler()₀ abstract ViewHandler getViewHandler()₀ abstract void setApplicationHandler(ApplicationHandler handler)₀</pre>	<p>Execute all of the phases of the request processing lifecycle, as described in the JavaServer Faces Specification, in the specified order.</p> <p>Execute the specified <code>Phase₀</code> as part of the request processing lifecycle being managed by this <code>Lifecycle</code> instance, by calling its <code>execute()</code> method.</p> <p>Return the <code>ApplicationHandler₀</code> instance that will be utilized during the <i>Invoke Application</i> phase of the request processing lifecycle.</p> <p>Return the <code>ViewHandler₀</code> instance that will be utilized during the <i>Render Response</i> phase of the request processing lifecycle.</p> <p>Set the <code>ApplicationHandler₀</code> instance that will be utilized during the <i>Invoke Application</i> phase of the request processing lifecycle.</p>
---	--

Member Summary

```
abstract void setViewHandler(ViewHandler handler)
```

Set the `ViewHandler` instance that will be utilized during the *Render Response* phase of the request processing lifecycle.

Inherited Member Summary**Methods inherited from class Object**

```
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(),
toString(), wait(), wait(), wait()
```

Constructors

Lifecycle()

```
public Lifecycle()
```

Methods

execute(FacesContext)

```
public abstract void execute(javax.faces.context.FacesContext context)
                            throws FacesException
```

Execute all of the phases of the request processing lifecycle, as described in the JavaServer Faces Specification, in the specified order. The processing flow can be affected (by the application, by components, or by event listeners) by calls to the `renderResponse()` or `responseComplete()` methods of the `javax.faces.context.FacesContext` instance associated with the current request.

Parameters:

`context` - FacesContext for the request to be processed

Throws:

`javax.faces.FacesException` - if thrown during the execution of the request processing lifecycle

`NullPointerException` - if `context` is null

executePhase(FacesContext, Phase)

```
public abstract int executePhase(javax.faces.context.FacesContext0 context,  
                                javax.faces.lifecycle.Phase0 phase)  
                                throws FacesException
```

Deprecated. The Phase₀ API will likely be removed in a future version of JavaServer Faces.

Execute the specified Phase₀ as part of the request processing lifecycle being managed by this Lifecycle instance, by calling its execute() method. Return the *state change indicator* value that was returned by the Phase₀.

Parameters:

context - FacesContext for the current request being processed

phase - Phase instance to be processed

Throws:

javax.faces.FacesException₀ - if thrown by the execute() method of the Phase₀ that was executed

java.lang.IllegalStateException - if a Phase₀ returned an invalid state change value from the execute() method

NullPointerException - if context or phase is null

getApplicationHandler()

```
public abstract javax.faces.lifecycle.ApplicationHandler0 getApplicationHandler()
```

Deprecated. The current mechanism for handling application events is a placeholder, and will be replaced in the next public release of JavaServer Faces.

Return the ApplicationHandler₀ instance that will be utilized during the *Invoke Application* phase of the request processing lifecycle.

getViewHandler()

```
public abstract javax.faces.lifecycle.ViewHandler0 getViewHandler()
```

Return the ViewHandler₀ instance that will be utilized during the *Render Response* phase of the request processing lifecycle.

setApplicationHandler(ApplicationHandler)

```
public abstract void  
setApplicationHandler(javax.faces.lifecycle.ApplicationHandler0  
handler)
```

Deprecated. The current mechanism for handling application events is a placeholder, and will be replaced in the next public release of JavaServer Faces.

Set the `ApplicationHandler0` instance that will be utilized during the *Invoke Application* phase of the request processing lifecycle.

Parameters:

`handler` - The new `ApplicationHandler0` instance

Throws:

`java.lang.IllegalStateException` - if this method is called after at least one request has been processed by this `Lifecycle` instance

`NullPointerException` - if `handler` is null

setViewHandler(ViewHandler)

```
public abstract void setViewHandler(javax.faces.lifecycle.ViewHandler0 handler)
```

Set the `ViewHandler0` instance that will be utilized during the *Render Response* phase of the request processing lifecycle.

Parameters:

`handler` - The new `ViewHandler0` instance

Throws:

`java.lang.IllegalStateException` - if this method is called after at least one request has been processed by this `Lifecycle` instance

`NullPointerException` - if `handler` is null

javax.faces.lifecycle LifecycleFactory

Declaration

```
public abstract class LifecycleFactory  
  
java.lang.Object  
|  
+--javax.faces.lifecycle.LifecycleFactory
```

Description

LifecycleFactory is a factory object that creates (if needed) and returns *Lifecycle₀* instances. Implementations of JavaServer Faces must provide at least a default implementation of *Lifecycle₀*. Advanced implementations (or external third party libraries) MAY provide additional *Lifecycle₀* implementations (keyed by lifecycle identifiers) for performing different types of request processing on a per-request basis.

There must be one *LifecycleFactory* instance per web application that is utilizing JavaServer Faces. This instance can be acquired, in a portable manner, by calling:

```
LifecycleFactory factory = (LifecycleFactory)  
    FactoryFinder.getFactory(FactoryFinder.LIFECYCLE_FACTORY);
```

Member Summary

Fields

static DEFAULT_LIFECYCLE ₀ java.lang.String	The lifecycle identifier for the default <i>Lifecycle₀</i> instance for this JavaServer Faces implementation.
---	--

Constructors

```
LifecycleFactory()
```

Methods

abstract void addLifecycle(java.lang.String lifecycleId, Lifecycle lifecycle) ₀	Register a new <i>Lifecycle₀</i> instance, associated with the specified lifecycleId, to be supported by this <i>LifecycleFactory</i> .
abstract Lifecycle getLifecycle(java.lang.String lifecycleId) ₀	Create (if needed) and return a <i>Lifecycle₀</i> instance for the specified lifecycle identifier.
abstract java.util.Iterator getLifecycleIds() ₀	Return an Iterator over the set of lifecycle identifiers supported by this factory.

Inherited Member Summary

Methods inherited from class `Object`

```
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(),
toString(), wait(), wait(), wait()
```

Fields

`DEFAULT_LIFECYCLE`

```
public static final java.lang.String DEFAULT_LIFECYCLE
```

The lifecycle identifier for the default `Lifecycle0` instance for this JavaServer Faces implementation.

Constructors

`LifecycleFactory()`

```
public LifecycleFactory()
```

Methods

`addLifecycle(String, Lifecycle)`

```
public abstract void addLifecycle(java.lang.String lifecycleId,
                                  javax.faces.lifecycle.Lifecycle0 lifecycle)
```

Register a new `Lifecycle0` instance, associated with the specified `lifecycleId`, to be supported by this `LifecycleFactory`. This method may be called at any time, and makes the corresponding `Lifecycle0` instance available throughout the remaining lifetime of this web application.

Parameters:

`lifecycleId` - Identifier of the new `Lifecycle0`

`lifecycle` - `Lifecycle0` instance that we are registering

Throws:

`java.lang.IllegalArgumentException` - if a `Lifecycle0` with the specified `lifecycleId` has already been registered

NullPointerException - if lifecycleId or lifecycle is null

getLifecycle(String)

```
public abstract javax.faces.lifecycle.Lifecycle0 getLifecycle(java.lang.String  
lifecycleId)
```

Create (if needed) and return a Lifecycle₀ instance for the specified lifecycle identifier. The set of available lifecycle identifiers is available via the `getLifecycleIds()` method.

Each call to `getLifecycle()` for the same lifecycleId, from within the same web application, must return the same Lifecycle₀ instance.

Parameters:

lifecycleId - Lifecycle identifier of the requested Lifecycle₀ instance

Throws:

java.lang.IllegalArgumentException - if no Lifecycle₀ instance can be returned for the specified identifier

NullPointerException - if lifecycleId is null

getLifecycleIds()

```
public abstract java.util.Iterator getLifecycleIds()
```

Return an Iterator over the set of lifecycle identifiers supported by this factory. This set must include the value specified by `LifecycleFactory.DEFAULT_LIFECYCLE`.

javax.faces.lifecycle Phase

Declaration

```
public abstract class Phase  
  
java.lang.Object  
|  
+--javax.faces.lifecycle.Phase
```

Description

Deprecated. The `Phase`₀ API will likely be removed in a future version of JavaServer Faces.

A `Phase` is a single step in the processing of a JavaServer Faces request throughout its entire `Lifecycle`₀. Each `Phase` performs the required transitions on the state information in the `javax.faces.context.FacesContext`₀ associated with this request, and returns one of the specified state change values to determine which `Phase` (if any) will be the next one to be processed.

Individual `Phase` instances must be instantiated and associated with a particular `Lifecycle`₀ instance in the `LifecycleFactory`₀ instance for this web application, prior to return from `getLifecycle()`. Each `Phase` instance will be associated with exactly one `Lifecycle`₀ instance, but must be programmed in a thread-safe manner because it will be used to process simultaneous requests on multiple threads.

Member Summary

Fields

<code>static int GOTO_EXIT₀</code>	Return value from <code>execute()</code> that terminates execution of the request processing lifecycle for this request.
<code>static int GOTO_NEXT₀</code>	Return value from <code>execute()</code> that requests execution of the next phase in order.
<code>static int GOTO_RENDER₀</code>	Return value from <code>execute()</code> that requests execution to skip intervening lifecycle phases and resume with the <i>Render Response</i> phase.

Constructors

`Phase()`₀

Methods

Member Summary

```
abstract int execute(javax.faces.context.FacesContext context)
    Perform all state transitions required by the current phase of the request processing
    Lifecycle0 for a particular request.
```

Inherited Member Summary**Methods inherited from class Object**

```
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(),
toString(), wait(), wait(), wait()
```

Fields

GOTO_EXIT

```
public static final int GOTO_EXIT
```

Return value from `execute()` that terminates execution of the request processing lifecycle for this request. This value will normally be returned only from the *Render Response* phase, but might also be returned in abnormal circumstances such as execution of an HTTP redirect. In all cases where this value is returned, the phase implementation must have guaranteed that the `ServletResponse` associated with this request has been completed.

GOTO_NEXT

```
public static final int GOTO_NEXT
```

Return value from `execute()` that requests execution of the next phase in order.

GOTO_RENDER

```
public static final int GOTO_RENDER
```

Return value from `execute()` that requests execution to skip intervening lifecycle phases and resume with the *Render Response* phase. Execution will start with the latest custom phase added (via `LifecycleFactory.registerBefore()`) for the *Render Response* phase for our lifecycle instance.

Constructors

Phase()

```
public Phase()
```

Methods

execute(FacesContext)

```
public abstract int execute(javax.faces.context.FacesContext context)
                           throws FacesException
```

Perform all state transitions required by the current phase of the request processing Lifecycle₀ for a particular request. Return one of the standard state change values (GOTO_EXIT, GOTO_NEXT, or GOTO_RENDER) to indicate what the request processing lifecycle should do next.<?p>

Parameters:

context - FacesContext for the current request being processed

Throws:

javax.faces.FacesException₀ - if a processing error occurred while executing this phase

javax.faces.lifecycle ViewHandler

Declaration

```
public interface ViewHandler
```

Description

ViewHandler is an interface defining a mechanism by which the *Render Response* phase of the request processing lifecycle can use a pluggable mechanism to support different response generation technologies. JSF implementations, or JSF-based applications, can register an instance of this interface by calling the `setViewHandler()` method of the `Lifecycle0` instance used to process incoming requests, prior to receiving the first request.

A default implementation of `ViewHandler` must be provided by the JSF implementation, which will be utilized unless `setViewHandler()` is called to establish a different one. This default instance will treat the `treeId` property of the response component tree as a context-relative path (after prefixing it with a slash), and will perform a `RequestDispatcher.forward()` call to that path.

Member Summary

Methods

```
void renderView(javax.faces.context.FacesContext context)
    Perform whatever actions are required to render the response component tree to the
    ServletResponse associated with the specified
    javax.faces.context.FacesContext0.
```

Methods

renderView(FacesContext)

```
public void renderView(javax.faces.context.FacesContext0 context)
    throws IOException, ServletException
```

Perform whatever actions are required to render the response component tree to the `ServletResponse` associated with the specified `javax.faces.context.FacesContext0`.

Parameters:

`context - javax.faces.context.FacesContext0` for the current request

Throws:

`java.io.IOException` - if an input/output error occurs
`NullPointerException` - if context is null
`javax.servlet.ServletException` - if a servlet error occurs

JSF.10.7 Package `javax.faces.render`

Package `javax.faces.render`

Description

Classes and interfaces defining the rendering model.

Class Summary	
Classes	
<code>Renderer</code>	A Renderer converts the internal representation of <code>javax.faces.component.UIComponent</code> s into the output stream (or writer) associated with the response we are creating for a particular request.
<code>RenderKit</code>	RenderKit represents a collection of <code>Renderer</code> instances that, together, know how to render JavaServer Faces <code>javax.faces.component.UIComponent</code> instances for a specific client.
<code>RenderKitFactory</code>	RenderKitFactory is a factory object that creates (if needed) and returns new <code>RenderKit</code> instances.

javax.faces.render Renderer

Declaration

```
public abstract class Renderer  
  
java.lang.Object  
|  
+--javax.faces.render.Renderer
```

Description

A **Renderer** converts the internal representation of `javax.faces.component.UIComponent`s into the output stream (or writer) associated with the response we are creating for a particular request. Each **Renderer** knows how to render one or more `javax.faces.component.UIComponent` types (or classes), and advertises a set of render-dependent attributes that it recognizes for each supported `javax.faces.component.UIComponent`.

Families of **Renderer**s are packaged as a `RenderKit`, and together support the rendering of all of the `javax.faces.component.UIComponent`s in the component tree associated with a `javax.faces.context.FacesContext`. Within the set of **Renderer**s for a particular `RenderKit`, each must be uniquely identified by the `rendererType` property.

Individual **Renderer** instances will be instantiated as requested during the rendering process, and will remain in existence for the remainder of the lifetime of a web application. Because each instance may be invoked from more than one request processing thread simultaneously, they **MUST** be programmed in a thread-safe manner.

Member Summary

Constructors

`Renderer()`

Methods

```
abstract void decode(javax.faces.context.FacesContext context,  
                      javax.faces.component.UIComponent component)  
          Decode the current state of the specified  
          javax.faces.component.UIComponent from the request contained in the  
          specified javax.faces.context.FacesContext, and attempt to convert  
          this state information into an object of the type required for this component (optionally  
          using the registered javax.faces.convert.Converter for this component,  
          if there is one).
```

Member Summary	
abstract void	encodeBegin(javax.faces.context.FacesContext context, javax.faces.component.UIComponent component) ₀ Render the beginning specified javax.faces.component.UIComponent ₀ to the output stream or writer associated with the response we are creating.
abstract void	encodeChildren(javax.faces.context.FacesContext context, javax.faces.component.UIComponent component) ₀ Render the child components of this javax.faces.component.UIComponent ₀ , following the rules described for encodeBegin() to acquire the appropriate value to be rendered.
abstract void	encodeEnd(javax.faces.context.FacesContext context, javax.faces.component.UIComponent component) ₀ Render the ending of the current state of the specified javax.faces.component.UIComponent ₀ , following the rules described for encodeBegin() to acquire the appropriate value to be rendered.
abstract javax.faces.component.AttributeDescriptor	getAttributeDescriptor(java.lang.String componentType, java.lang.String name) ₀ Return an javax.faces.component.AttributeDescriptor ₀ for the specified attribute name, as supported for the specified component type.
abstract javax.faces.component.AttributeDescriptor	getAttributeDescriptor(javax.faces.component.UIComponent component, java.lang.String name) ₀ Return an javax.faces.component.AttributeDescriptor ₀ for the specified attribute name, as supported for the specified component class.
abstract java.util.Iterator	getAttributeNames(java.lang.String componentType) ₀ Return an Iterator over the names of the supported attributes for the specified component type.
abstract java.util.Iterator	getAttributeNames(javax.faces.component.UIComponent component) ₀ Return an Iterator over the names of the supported attributes for the specified javax.faces.component.UIComponent ₀ implementation class.
abstract java.lang.String	getClientId(javax.faces.context.FacesContext context, javax.faces.component.UIComponent component) ₀ Return a client-side identifier for the argument component.
abstract boolean	supportsComponentType(java.lang.String componentType) ₀ Return true if this Renderer supports components of the specified type.
abstract boolean	supportsComponentType(javax.faces.component.UIComponent component) ₀ Return true if this Renderer supports components of the specified javax.faces.component.UIComponent ₀ class.

Inherited Member Summary

Methods inherited from class Object

Inherited Member Summary

```
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(),
toString(), wait(), wait(), wait()
```

Constructors

Renderer()

```
public Renderer()
```

Methods

decode(FacesContext, UIComponent)

```
public abstract void decode(javax.faces.context.FacesContext0 context,
                           javax.faces.component.UIComponent0 component)
                           throws IOException
```

Decode the current state of the specified `javax.faces.component.UIComponent0` from the request contained in the specified `javax.faces.context.FacesContext0`, and attempt to convert this state information into an object of the type required for this component (optionally using the registered `javax.faces.convert.Converter0` for this component, if there is one).

If conversion is successful:

- Save the new local value of this component by calling `setValue()` and passing the new value.
- Set the `value` property of this component to `true`.

If conversion is not successful:

- Save the state information (inside the component) in such a way that encoding can reproduce the previous input (even though it was syntactically or semantically incorrect).
- Add an appropriate conversion failure error message by calling `addMessage()` on the specified `javax.faces.context.FacesContext0`.
- Set the `valid` property of this component to `false`.

During decoding, events may be enqueued for later processing (by event listeners that have registered an interest), by calling `addFacesEvent()` on the associated `javax.faces.context.FacesContext0`.

Parameters:

context - javax.faces.context.FacesContext₀ for the request we are processing
component - javax.faces.component.UIComponent₀ to be decoded.

Throws:

java.io.IOException - if an input/output error occurs while decoding
NullPointerException - if context or component is null

encodeBegin(FacesContext, UIComponent)

```
public abstract void encodeBegin(javax.faces.context.FacesContext0 context,  
                                javax.faces.component.UIComponent0 component)  
throws IOException
```

Render the beginning specified javax.faces.component.UIComponent₀ to the output stream or writer associated with the response we are creating. If the conversion attempted in a previous call to decode for this component failed, the state information saved during execution of decode() should be utilized to reproduce the incorrect input. If the conversion was successful, or if there was no previous call to decode(), the value to be displayed should be acquired by calling component.currentValue(), and rendering the value as appropriate.

Parameters:

context - javax.faces.context.FacesContext₀ for the request we are processing
component - javax.faces.component.UIComponent₀ to be rendered

Throws:

java.io.IOException - if an input/output error occurs while rendering
NullPointerException - if context or component is null

encodeChildren(FacesContext, UIComponent)

```
public abstract void encodeChildren(javax.faces.context.FacesContext0 context,  
                                    javax.faces.component.UIComponent0 component)  
throws IOException
```

Render the child components of this javax.faces.component.UIComponent₀, following the rules described for encodeBegin() to acquire the appropriate value to be rendered. This method will only be called if the rendersChildren property of this component is true.

Parameters:

context - javax.faces.context.FacesContext₀ for the response we are creating
component - javax.faces.component.UIComponent₀ whose children are to be rendered

Throws:

java.io.IOException - if an input/output error occurs while rendering
NullPointerException - if context or component is null

encodeEnd(FacesContext, UIComponent)

```
public abstract void encodeEnd(javax.faces.context.FacesContext0 context,  
                               javax.faces.component.UIComponent0 component)  
                           throws IOException
```

Render the ending of the current state of the specified
javax.faces.component.UIComponent₀, following the rules described for
encodeBegin() to acquire the appropriate value to be rendered.

Parameters:

context - javax.faces.context.FacesContext₀ for the response we are creating
component - javax.faces.component.UIComponent₀ to be rendered

Throws:

java.io.IOException - if an input/output error occurs while rendering
NullPointerException - if context or component is null

getAttributeDescriptor(String, String)

```
public abstract javax.faces.component.AttributeDescriptor0  
                  getAttributeDescriptor(java.lang.String componentType,  
                                         java.lang.String name)
```

Return an javax.faces.component.AttributeDescriptor₀ for the specified attribute
name, as supported for the specified component type.

Parameters:

componentType - Canonical name of a supported component type
name - Name of the attribute for which to return a descriptor

Throws:

java.lang.IllegalArgumentException - if the specified component type is not
supported
java.lang.IllegalArgumentException - if the specified attribute name is not
supported for the specified component type
NullPointerException - if type or name is null

getAttributeDescriptor(UIComponent, String)

```
public abstract javax.faces.component.AttributeDescriptor0
    getAttributeDescriptor(javax.faces.component.UIComponent0 component,
        java.lang.String name)
```

Return an `javax.faces.component.AttributeDescriptor0` for the specified attribute name, as supported for the specified component class.

Parameters:

component - `javax.faces.component.UIComponent0` whose implementation class is supported

name - Name of the attribute for which to return a descriptor

Throws:

`java.lang.IllegalArgumentException` - if the specified component class is not supported

`java.lang.IllegalArgumentException` - if the specified attribute name is not supported for the specified component class

`NullPointerException` - if component or name is null

getAttributeNames(String)

```
public abstract java.util.Iterator getAttributeNames(java.lang.String
    componentType)
```

Return an `Iterator` over the names of the supported attributes for the specified component type. If no attributes are supported, an empty `Iterator` is returned.

Parameters:

componentType - Canonical name of a supported component type

Throws:

`java.lang.IllegalArgumentException` - if the specified component type is not supported

`NullPointerException` - if type is null

getAttributeNames(UIComponent)

```
public abstract java.util.Iterator
    getAttributeNames(javax.faces.component.UIComponent0 component)
```

Return an `Iterator` over the names of the supported attributes for the specified `javax.faces.component.UIComponent0` implementation class. If no attributes are supported, an empty `Iterator` is returned.

Parameters:

component - javax.faces.component.UIComponent_o whose implementation class is supported

Throws:

java.lang.IllegalArgumentException - if the specified component class is not supported

NullPointerException - if component is null

getClientId(FacesContext, UIComponent)

```
public abstract java.lang.String getClientId(javax.faces.context.FacesContexto, context, javax.faces.component.UIComponento, component)
```

Return a client-side identifier for the argument component.

If a client-side identifier has previously been generated for this component, and saved in the attribute named by UIComponent.CLIENT_ID, return that identifier value. Otherwise, generate a new client-side identifier, save it in the attribute named by UIComponent.CLIENT_ID on the specified javax.faces.component.UIComponent_o, and return it.

Parameters:

context - javax.faces.context.FacesContext_o for the current request

component - javax.faces.component.UIComponent_o whose identifier is to be returned

Throws:

NullPointerException - if context or component is null

supportsComponentType(String)

```
public abstract boolean supportsComponentType(java.lang.String componentType)
```

Return true if this Renderer supports components of the specified type. This is matched against the type property of each javax.faces.component.UIComponent_o to determine compatibility.

Parameters:

componentType - Canonical name of the component type to be tested for compatibility

Throws:

NullPointerException - if componentType is null

supportsComponentType(UIComponent)

```
public abstract boolean supportsComponentType(javax.faces.component.UIComponento, component)
```

Return `true` if this `Renderer` supports components of the specified `javax.faces.component.UIComponent0` class.

Parameters:

`component` - `javax.faces.component.UIComponent0` whose implementation class is checked for compatibility

Throws:

`NullPointerException` - if `component` is `null`

javax.faces.render RenderKit

Declaration

```
public abstract class RenderKit  
  
java.lang.Object  
|  
+--javax.faces.render.RenderKit
```

Description

RenderKit represents a collection of *Renderer*₀ instances that, together, know how to render JavaServer Faces `javax.faces.component.UIComponent`₀ instances for a specific client. Typically, `RenderKit`₀s are specialized for some combination of client device type, markup language, and/or user Locale. A `RenderKit`₀ also acts as a Factory for associated *Renderer*₀ instances, which perform the actual rendering process for each component.

A typical JavaServer Faces implementation will configure one or more `RenderKit`₀ instances at web application startup. They are made available through calls to the `getRenderKit()` methods of `RenderKitFactory`₀. Because `RenderKit`₀ instances are shared, they must be implemented in a thread-safe manner.

Member Summary

Constructors

```
RenderKit()
```

Methods

<pre>abstract void addComponentClass(java.lang.Class componentClass)</pre>	Add a new <code>javax.faces.component.UIComponent</code> ₀ subclass to the set of component classes registered with this <code>RenderKit</code> ₀ instance and supported by its constituent <i>Renderer</i> ₀ s.
<pre>abstract void addRenderer(java.lang.String rendererType, Renderer renderer)</pre>	Add a new <i>Renderer</i> ₀ instance, associated with the specified <code>rendererType</code> , to the set of <i>Renderer</i> ₀ s registered with this <code>RenderKit</code> ₀ .
<pre>abstract java.util.Iterator getComponentClasses()</pre>	Return an <code>Iterator</code> of the <code>javax.faces.component.UIComponent</code> ₀ classes that are registered with this <code>RenderKit</code> ₀ instance and supported by its constituent <i>Renderer</i> ₀ s.

Member Summary

```
abstract Renderer getRenderer(java.lang.String rendererType)
    Create (if necessary) and return a Renderer0 instance with the specified renderer
    type.

abstract getRendererTypes()
java.util.Iterator
    Return an Iterator of all the renderer types registered with this RenderKit.

abstract getRendererTypes(java.lang.String componentType)
java.util.Iterator
    Return an Iterator of the renderer types of all Renderer0 instances registered
    with this RenderKit0 that support components of the specified component type.

abstract getRendererTypes(javax.faces.component.UIComponent
java.util.Iterator component)
    Return an Iterator of the renderer types of all Renderer0 instances registered
    with this RenderKit0 that support components of the specified
    javax.faces.component.UIComponent0 subclass.
```

Inherited Member Summary

Methods inherited from class Object

```
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(),
toString(), wait(), wait(), wait()
```

Constructors

RenderKit()

```
public RenderKit()
```

Methods

addComponentClass(Class)

```
public abstract void addComponentClass(java.lang.Class componentClass)
```

Add a new javax.faces.component.UIComponent₀ subclass to the set of component classes registered with this RenderKit₀ instance and supported by its constituent Renderer₀s.

Parameters:

componentClass - javax.faces.component.UIComponent₀ subclass to be supported

Throws:

`java.lang.IllegalArgumentException` - if `componentClass` is not a
`javax.faces.component.UIComponent` subclass
`NullPointerException` - if `componentClass` is null

addRenderer(String, Renderer)

```
public abstract void addRenderer(java.lang.String rendererType,  
                                 javax.faces.render.Renderer renderer)
```

Add a new `Renderer` instance, associated with the specified `rendererType`, to the set of `Renderer`s registered with this `RenderKit`.

Parameters:

`rendererType` - Renderer type of the new `Renderer`
`renderer` - The new `Renderer` instance

Throws:

`java.lang.IllegalArgumentException` - if a `Renderer` with the specified `rendererType` has already been registered
`NullPointerException` - if `rendererType` or `renderer` is null

getComponentClasses()

```
public abstract java.util.Iterator getComponentClasses()
```

Return an `Iterator` of the `javax.faces.component.UIComponent` classes that are registered with this `RenderKit` instance and supported by its constituent `Renderer`s. If there are no such registered component classes, an empty `Iterator` is returned.

getRenderer(String)

```
public abstract javax.faces.render.Renderer getRenderer(java.lang.String  
                                                       rendererType)
```

Create (if necessary) and return a `Renderer` instance with the specified renderer type. Subsequent calls to this method with the same `rendererType`, from the same web application, must return the same instance.

Parameters:

`rendererType` - Renderer type to be returned

Throws:

`java.lang.IllegalArgumentException` - if the requested renderer type is not supported by this `RenderKit`
`NullPointerException` - if `rendererType` is null

getRendererTypes()

```
public abstract java.util.Iterator getRendererTypes()
```

Return an `Iterator` of all the renderer types registered with this `RenderKit`.

getRendererTypes(String)

```
public abstract java.util.Iterator getRendererTypes(java.lang.String  
    componentType)
```

Return an `Iterator` of the renderer types of all `Renderer`_o instances registered with this `RenderKit`_o that support components of the specified component type. If no `Renderer`_os support this component type, an empty `Iterator` is returned.

Parameters:

`componentType` - Canonical name of the component type tested for support

Throws:

`NullPointerException` - if `componentType` is null

getRendererTypes(UIComponent)

```
public abstract java.util.Iterator  
    getRendererTypes(javax.faces.component.UIComponent component)
```

Return an `Iterator` of the renderer types of all `Renderer`_o instances registered with this `RenderKit`_o that support components of the specified `javax.faces.component.UIComponent`_o subclass. If no `Renderer`_os support this component class, an empty `Iterator` is returned.

Parameters:

`component` - `UIComponent` whose class is tested for support

Throws:

`NullPointerException` - if `component` is null

javax.faces.render RenderKitFactory

Declaration

```
public abstract class RenderKitFactory  
  
java.lang.Object  
|  
+--javax.faces.render.RenderKitFactory
```

Description

RenderKitFactory is a factory object that creates (if needed) and returns new *RenderKit*₀ instances. Implementations of JavaServer Faces must provide at least a default implementation of *RenderKit*₀. Advanced implementations (or external third party libraries) may provide additional *RenderKit*₀ implementations (keyed by render kit identifiers) for performing different types of rendering for the same components.

There must be one *RenderKitFactory*₀ instance per web application that is utilizing JavaServer Faces. This instance can be acquired, in a portable manner, by calling:

```
RenderKitFactory factory = (RenderKitFactory)  
    FactoryFinder.getFactory(FactoryFinder.RENDER_KIT_FACTORY);
```

Member Summary

Fields

static <code>DEFAULT_RENDER_KIT₀</code> <code>java.lang.String</code>	The render kit identifier of the default <i>RenderKit</i> ₀ instance for this JavaServer Faces implementation.
---	---

Constructors

```
RenderKitFactory()
```

Methods

abstract void <code>addRenderKit(java.lang.String renderKitId, RenderKit renderKit)</code>	Register a new <i>RenderKit</i> ₀ instance, associated with the specified <i>renderKitId</i> , to be supported by this <i>RenderKitFactory</i> ₀ .
abstract <i>RenderKit</i> <code>getRenderKit(java.lang.String renderKitId)</code>	Create (if needed) and return a <i>RenderKit</i> ₀ instance for the specified render kit identifier.

Member Summary	
abstract RenderKit <code>getRenderKit(java.lang.String renderKitId, javax.faces.context.FacesContext context)</code>	Create (if needed) and return a <code>RenderKit</code> , instance for the specified render kit identifier, possibly customized based on dynamic characteristics of the specified <code>javax.faces.context.FacesContext</code> .
abstract <code>getRenderKitIds()</code> , java.util.Iterator	Return an Iterator over the set of render kit identifiers supported by this factory.

Inherited Member Summary	
Methods inherited from class Object	
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()	

Fields

DEFAULT_RENDER_KIT

```
public static final java.lang.String DEFAULT_RENDER_KIT
```

The render kit identifier of the default `RenderKit`, instance for this JavaServer Faces implementation.

Constructors

RenderKitFactory()

```
public RenderKitFactory()
```

Methods

addRenderKit(String, RenderKit)

```
public abstract void addRenderKit(java.lang.String renderKitId,  
                                 javax.faces.render.RenderKit renderKit)
```

Register a new `RenderKit`, instance, associated with the specified `renderKitId`, to be supported by this `RenderKitFactory`. This method may be called at any time, and makes the

corresponding `RenderKit` instance available throughout the remaining lifetime of this web application.

Parameters:

`renderKitId` - Identifier of the new `RenderKit`
`renderKit` - `RenderKit` instance that we are registering

Throws:

`java.lang.IllegalArgumentException` - if a `RenderKit` with the specified `renderKitId` has already been registered
`NullPointerException` - if `renderKitId` or `renderKit` is null

getRenderKit(String)

```
public abstract javax.faces.render.RenderKit getRenderKit(java.lang.String renderKitId)
```

Create (if needed) and return a `RenderKit` instance for the specified render kit identifier. The set of available render kit identifiers is available via the `getRenderKitIds()` method.

Each call to `getRenderKit()` for the same `renderKitId`, from within the same web application, must return the same `RenderKit` instance.

Parameters:

`renderKitId` - Render kit identifier of the requested `RenderKit` instance

Throws:

`java.lang.IllegalArgumentException` - if no `RenderKit` instance can be returned for the specified identifier
`NullPointerException` - if `renderKitId` is null

getRenderKit(String, FacesContext)

```
public abstract javax.faces.render.RenderKit getRenderKit(java.lang.String renderKitId, javax.faces.context.FacesContext context)
```

Create (if needed) and return a `RenderKit` instance for the specified render kit identifier, possibly customized based on dynamic characteristics of the specified `javax.faces.context.FacesContext`. The set of available render kit identifiers is available via the `getRenderKitIds()` method.

Parameters:

`renderKitId` - Render kit identifier of the requested `RenderKit` instance
`context` - `FacesContext` for the request currently being processed

Throws:

`java.lang.IllegalArgumentException` - if no `RenderKit` instance can be returned for the specified identifier

`NullPointerException` - if `renderKitId` or `context` is null

getRenderKitIds()

```
public abstract java.util.Iterator getRenderKitIds()
```

Return an `Iterator` over the set of render kit identifiers supported by this factory. This set must include the value specified by `RenderKitFactory.DEFAULT_RENDER_KIT`.

JSF.10.8 Package javax.faces.tree

Package **javax.faces.tree**

Description

Classes and interfaces defining the component tree and component tree factory.

Class Summary

Classes

`Tree0`

`Tree` is a container for a tree of `javax.faces.component.UIComponent0` nodes rooted at a root node, along with associated metadata about those `javax.faces.component.UIComponent0`s.

`TreeFactory0`

`TreeFactory` is a factory object that creates and returns new `Tree0` instances.

javax.faces.tree Tree

Declaration

```
public abstract class Tree implements java.io.Serializable  
  
java.lang.Object  
|  
+--javax.faces.tree.Tree
```

All Implemented Interfaces: java.io.Serializable

Description

Tree is a container for a tree of `javax.faces.component.UIComponent` nodes rooted at a root node, along with associated metadata about those `javax.faces.component.UIComponent`s. Trees have unique identifiers, which may associate them with metadata information that can be acquired from external sources, by means of a `TreeFactory` instance. Tree identifiers must be unique within the scope of a web application.

Member Summary

Constructors

`Tree()`

Methods

abstract	<code>getRenderKitId()</code>	
<code>java.lang.String</code>		Return the render kit identifier of the <code>javax.faces.render.RenderKit</code> instance (if any) associated with this <code>Tree</code> .
abstract	<code>getRoot()</code>	Return the root node of the component tree associated with this <code>Tree</code> .
<code>javax.faces.component.UIComponent</code>		
abstract	<code>getTreeId()</code>	Return the tree identifier for this <code>Tree</code> .
<code>java.lang.String</code>		
abstract void	<code>setRenderKitId(java.lang.String renderKitId)</code>	Set the render kit identifier of the <code>javax.faces.render.RenderKit</code> instance (if any) associated with this <code>Tree</code> .

Inherited Member Summary

Methods inherited from class `Object`

`clone()`, `equals(Object)`, `finalize()`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`,
`toString()`, `wait()`, `wait()`, `wait()`

Constructors

`Tree()`

```
public Tree()
```

Methods

`getRenderKitId()`

```
public abstract java.lang.String getRenderKitId()
```

Return the render kit identifier of the `javax.faces.render.RenderKito` instance (if any) associated with this `Treeo`.

`getRoot()`

```
public abstract javax.faces.component.UIComponento getRoot()
```

Return the root node of the component tree associated with this `Treeo`.

`getTreeId()`

```
public abstract java.lang.String getTreeId()
```

Return the tree identifier for this `Treeo`.

`setRenderKitId(String)`

```
public abstract void setRenderKitId(java.lang.String renderKitId)
```

Set the render kit identifier of the `javax.faces.render.RenderKito` instance (if any) associated with this `Treeo`.

Parameters:

`renderKitId` - The new `javax.faces.render.RenderKito` identifier, or null to disassociate this tree with any specific `javax.faces.render.RenderKito` instance

javax.faces.tree TreeFactory

Declaration

```
public abstract class TreeFactory  
  
java.lang.Object  
|  
+--javax.faces.tree.TreeFactory
```

Description

TreeFactory is a factory object that creates and returns new *Tree*₀ instances.

There must be one *TreeFactory*₀ instance per web application that is utilizing JavaServer Faces. This instance can be acquired, in a portable manner, by calling:

```
TreeFactory factory = (TreeFactory)  
    FactoryFinder.getFactory(FactoryFinder.TREE_FACTORY);
```

Member Summary

Constructors

```
TreeFactory()0
```

Methods

```
abstract Tree getTree(javax.faces.context.FacesContext context,  
                      java.lang.String treeId)0  
Create (if needed) and return a Tree0 that is initialized with a root node, and whose  
renderKitId property is initialized to the identifier of the default  
javax.faces.render.RenderKit0.
```

Inherited Member Summary

Methods inherited from class Object

```
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(),  
toString(), wait(), wait(), wait()
```

Constructors

TreeFactory()

```
public TreeFactory()
```

Methods

getTree(FacesContext, String)

```
public abstract javax.faces.tree.Tree0 getTree(javax.faces.context.FacesContext0,  
        context, java.lang.String treeId)  
        throws FacesException
```

Create (if needed) and return a Tree₀ that is initialized with a root node, and whose renderKitId property is initialized to the identifier of the default javax.faces.render.RenderKit₀.

Parameters:

context - javax.faces.context.FacesContext₀ for the current request
treeId - Tree identifier of the tree to be constructed and returned

Throws:

javax.faces.FacesException₀ - if a Tree₀ cannot be constructed for the specified parameters

NullPointerException - if context or treeId is null

JSF.10.9 Package `javax.faces.validator`

Package `javax.faces.validator`

Description

Interface defining the validator model, and concrete validator implementation classes.

Class Summary	
Interfaces	
<code>Validator</code>	A Validator implementation is a class that can perform validation (correctness checks) on a <code>javax.faces.component.UIComponent</code> .
Classes	
<code>DoubleRangeValidator</code>	DoubleRangeValidator is a <code>Validator</code> that checks the value of the corresponding component against specified minimum and maximum values.
<code>LengthValidator</code>	LengthValidator is a <code>Validator</code> that checks the number of characters in the String representation of the value of the associated component.
<code>LongRangeValidator</code>	LongRangeValidator is a <code>Validator</code> that checks the value of the corresponding component against specified minimum and maximum values.
<code>RequiredValidator</code>	RequiredValidator is a <code>Validator</code> that checks for the existence of a value for the associated component.
<code>StringRangeValidator</code>	StringRangeValidator is a <code>Validator</code> that checks the value of the corresponding component against specified minimum and maximum values.

`javax.faces.validator`

DoubleRangeValidator

Declaration

```
public class DoubleRangeValidator extends ValidatorBase  
  
java.lang.Object  
|  
+--javax.faces.validator.ValidatorBase  
|  
+--javax.faces.validator.DoubleRangeValidator
```

All Implemented Interfaces: `java.io.Serializable, Validator`

Description

DoubleRangeValidator is a `Validator`, that checks the value of the corresponding component against specified minimum and maximum values. The following algorithm is implemented:

- Call `getValue()` to retrieve the current value of the component. If it is `null`, exit immediately. (If `null` values should not be allowed, a `RequiredValidator` can be configured to check for this case.)
- If the current component value is not a floating point type, or a `String` that is convertible to double, add a `TYPE_MESSAGE_ID` message to the `javax.faces.context.FacesContext` for this request, and skip subsequent checks.
- If a `maximum` property has been configured on this `Validator`, check the component value against this limit. If the component value is greater than the specified minimum, add a `MAXIMUM_MESSAGE_ID` message to the `javax.faces.context.FacesContext` for this request.
- If a `minimum` property has been configured on this `Validator`, check the component value against this limit. If the component value is less than the specified minimum, add a `MINIMUM_MESSAGE_ID` message to the `javax.faces.context.FacesContext` for this request.

Member Summary
Fields

Member Summary

static	MAXIMUM_MESSAGE_ID ₀	
java.lang.String	The message identifier of the javax.faces.context.Message ₀ to be created if the maximum value check fails.	
static	MINIMUM_MESSAGE_ID ₀	
java.lang.String	The message identifier of the javax.faces.context.Message ₀ to be created if the minimum value check fails.	
static	TYPE_MESSAGE_ID ₀	
java.lang.String	The message identifier of the javax.faces.context.Message ₀ to be created if the current value of this component is not of the correct type.	

Constructors

DoubleRangeValidator()	
	Construct a Validator ₀ with no preconfigured limits.
DoubleRangeValidator(double maximum)	
	Construct a Validator ₀ with the specified preconfigured limit.
DoubleRangeValidator(double maximum, double minimum)	
	Construct a Validator ₀ with the specified preconfigured limits.

Methods

protected	double	doubleValue(java.lang.Object attributeValue)	
			Return the specified attribute value, converted to a double.
	double	getMaximum()	
			Return the maximum value to be enforced by this Validator ₀ , if isMaximumSet() returns true.
protected	javax.faces.context.Message	getMessage(javax.faces.context.FacesContext context, java.lang.String messageId)	
			Return a javax.faces.context.Message ₀ for the specified parameters.
protected	javax.faces.context.Message	getMessage(javax.faces.context.FacesContext context, java.lang.String messageId, java.lang.Object params)	
			Return a javax.faces.context.Message ₀ for the specified parameters.
protected	javax.faces.context.MessageResources	getMessageResources()	
			Return the javax.faces.context.MessageResources ₀ instance for the message resources defined by the JavaServer Faces Specification.
	double	getMinimum()	
			Return the minimum value to be enforced by this Validator ₀ , if isMinimumSet() returns true.
	boolean	isMaximumSet()	
			Return a flag indicating whether a maximum limit has been set.
	boolean	isMinimumSet()	
			Return a flag indicating whether a minimum limit has been set.
protected	long	longValue(java.lang.Object attributeValue)	
			Return the specified attribute value, converted to a long.
	void	setMaximum(double maximum)	
			Set the maximum value to be enforced by this Validator ₀ .
	void	setMinimum(double minimum)	
			Set the minimum value to be enforced by this Validator ₀ .

Member Summary

```
protected stringValue(java.lang.Object attributeValue)
java.lang.String      Return the specified attribute value, converted to a String.
void validate(javax.faces.context.FacesContext context,
               javax.faces.component.UIComponent component)_o
```

Inherited Member Summary

Methods inherited from class `Object`

```
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(),
toString(), wait(), wait(), wait()
```

Fields

`MAXIMUM_MESSAGE_ID`

```
public static final java.lang.String MAXIMUM_MESSAGE_ID
```

The message identifier of the `javax.faces.context.Message` to be created if the maximum value check fails. The message format string for this message may optionally include a { 0 } placeholder, which will be replaced by the configured maximum value.

`MINIMUM_MESSAGE_ID`

```
public static final java.lang.String MINIMUM_MESSAGE_ID
```

The message identifier of the `javax.faces.context.Message` to be created if the minimum value check fails. The message format string for this message may optionally include a { 0 } placeholder, which will be replaced by the configured minimum value.

`TYPE_MESSAGE_ID`

```
public static final java.lang.String TYPE_MESSAGE_ID
```

The message identifier of the `javax.faces.context.Message` to be created if the current value of this component is not of the correct type.

Constructors

`DoubleRangeValidator()`

```
public DoubleRangeValidator()
```

Construct a Validator₀ with no preconfigured limits.

DoubleRangeValidator(double)

```
public DoubleRangeValidator(double maximum)
```

Construct a Validator₀ with the specified preconfigured limit.

Parameters:

maximum - Maximum value to allow

DoubleRangeValidator(double, double)

```
public DoubleRangeValidator(double maximum, double minimum)
```

Construct a Validator₀ with the specified preconfigured limits.

Parameters:

maximum - Maximum value to allow

minimum - Minimum value to allow

Throws:

java.lang.IllegalArgumentException - if a specified maximum value is less than a specified minimum value

Methods

doubleValue(Object)

```
protected double doubleValue(java.lang.Object attributeValue)
    throws NumberFormatException
```

Return the specified attribute value, converted to a double.

Parameters:

attributeValue - The attribute value to be converted

Throws:

java.lang.NumberFormatException - if conversion is not possible

getMaximum()

```
public double getMaximum()
```

Return the maximum value to be enforced by this Validator₀, if isMaximumSet() returns true.

getMessage(FacesContext, String)

```
protected javax.faces.context.Message0
    getMessage(javax.faces.context.FacesContext0 context,
        java.lang.String messageId)
```

Return a javax.faces.context.Message₀ for the specified parameters.

Parameters:

context - The javax.faces.context.FacesContext₀ associated with the request being processed

messageId - Message identifier of the requested message

getMessage(FacesContext, String, Object[])

```
protected javax.faces.context.Message0
    getMessage(javax.faces.context.FacesContext0 context,
        java.lang.String messageId, java.lang.Object[] params)
```

Return a javax.faces.context.Message₀ for the specified parameters.

Parameters:

context - The javax.faces.context.FacesContext₀ associated with the request being processed

messageId - Message identifier of the requested message

params - Substitution parameters for this message

getMessageResources()

```
protected javax.faces.context.MessageResources0 getMessageResources()
```

Return the javax.faces.context.MessageResources₀ instance for the message resources defined by the JavaServer Faces Specification.

getMinimum()

```
public double getMinimum()
```

Return the minimum value to be enforced by this Validator₀, if isMinimumSet() returns true.

isMaximumSet()

```
public boolean isMaximumSet()
```

Return a flag indicating whether a maximum limit has been set.

isMinimumSet()

```
public boolean isMinimumSet()
```

Return a flag indicating whether a minimum limit has been set.

longValue(Object)

```
protected long longValue(java.lang.Object attributeValue)
    throws NumberFormatException
```

Return the specified attribute value, converted to a long.

Parameters:

attributeValue - The attribute value to be converted

Throws:

java.lang.NumberFormatException - if conversion is not possible

setMaximum(double)

```
public void setMaximum(double maximum)
```

Set the maximum value to be enforced by this Validator_o.

Parameters:

maximum - The new maximum value

Throws:

java.lang.IllegalArgumentException - if a specified maximum value is less than a specified minimum value

setMinimum(double)

```
public void setMinimum(double minimum)
```

Set the minimum value to be enforced by this Validator_o.

Parameters:

minimum - The new minimum value

Throws:

java.lang.IllegalArgumentException - if a specified maximum value is less than a specified minimum value

stringValue(Object)

```
protected java.lang.String stringValue(java.lang.Object attributeValue)
```

Return the specified attribute value, converted to a String.

Parameters:

attributeValue - The attribute value to be converted

validate(FacesContext, UIComponent)

```
public void validate(javax.faces.context.FacesContext0 context,  
                     javax.faces.component.UIComponent0 component)
```

Description copied from interface: javax.faces.validator.Validator₀

Perform the correctness checks implemented by this Validator₀ against the specified javax.faces.component.UIComponent₀. If any violations are found:

- Add zero or more javax.faces.context.Message₀s to the specified javax.faces.context.FacesContext₀, specifying this javax.faces.component.UIComponent₀ as associated with the message, describing the nature of the violation(s) encountered.
- Set the valid property on the specified javax.faces.component.UIComponent₀ to false.

Overrides: validate in class ValidatorBase

javax.faces.validator LengthValidator

Declaration

```
public class LengthValidator extends ValidatorBase  
  
java.lang.Object  
|  
+--javax.faces.validator.ValidatorBase  
|  
+--javax.faces.validator.LengthValidator
```

All Implemented Interfaces: `java.io.Serializable, Validator`

Description

LengthValidator is a `Validator`, that checks the number of characters in the String representation of the value of the associated component. The following algorithm is implemented:

- Call `getValue()` to retrieve the current value of the component. If it is `null`, exit immediately. (If `null` values should not be allowed, a `RequiredValidator` can be configured to check for this case.)
- Convert the value to a String, if necessary, by calling its `toString()` method.
- If a `maximum` property has been configured on this `Validator`, check the length of the converted String against this limit. If the String length is larger than the specified minimum, add a `MAXIMUM_MESSAGE_ID` message to the `javax.faces.context.FacesContext` for this request.
- If a `minimum` property has been configured on this `Validator`, check the length of the converted String against this limit. If the String length is less than the specified minimum, add a `MINIMUM_MESSAGE_ID` message to the `javax.faces.context.FacesContext` for this request.

Member Summary

Fields

<code>static MAXIMUM_MESSAGE_ID</code>	
<code>java.lang.String</code>	The message identifier of the <code>javax.faces.context.Message</code> to be created if the maximum length check fails.

Member Summary

static MINIMUM_MESSAGE_ID₀
java.lang.String The message identifier of the javax.faces.context.Message₀ to be created if the minimum length check fails.

Constructors

LengthValidator()₀
Construct a Validator₀ with no preconfigured limits.
LengthValidator(int maximum)₀
Construct a Validator₀ with the specified preconfigured limit.
LengthValidator(int maximum, int minimum)₀
Construct a Validator₀ with the specified preconfigured limits.

Methods

protected double doubleValue(java.lang.Object attributeValue)
Return the specified attribute value, converted to a double.
int getMaximum()₀
Return the maximum length to be enforced by this Validator₀, if isMaximumSet() returns true.
protected getMessage(javax.faces.context.FacesContext context, java.lang.String messageId)
Return a javax.faces.context.Message₀ for the specified parameters.
protected getMessage(javax.faces.context.FacesContext context, java.lang.String messageId, java.lang.Object params)
Return a javax.faces.context.Message₀ for the specified parameters.
protected getMessageResources()
Return the javax.faces.context.MessageResources₀ instance for the message resources defined by the JavaServer Faces Specification.
int getMinimum()₀
Return the minimum length to be enforced by this Validator₀, if isMinimumSet() returns true.
boolean isMaximumSet()₀
Return a flag indicating whether a maximum length has been set.
boolean isMinimumSet()₀
Return a flag indicating whether a minimum limit has been set.
protected long longValue(java.lang.Object attributeValue)
Return the specified attribute value, converted to a long.
void setMaximum(int maximum)₀
Set the maximum length to be enforced by this Validator₀.
void setMinimum(int minimum)₀
Set the minimum length to be enforced by this Validator₀.
protected stringValue(java.lang.Object attributeValue)
java.lang.String Return the specified attribute value, converted to a String.
void validate(javax.faces.context.FacesContext context, javax.faces.component.UIComponent component)₀

Inherited Member Summary

Methods inherited from class Object

`clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(),
toString(), wait(), wait(), wait()`

Fields

MAXIMUM_MESSAGE_ID

`public static final java.lang.String MAXIMUM_MESSAGE_ID`

The message identifier of the `javax.faces.context.Message`₀ to be created if the maximum length check fails. The message format string for this message may optionally include a {0} placeholder, which will be replaced by the configured maximum length.

MINIMUM_MESSAGE_ID

`public static final java.lang.String MINIMUM_MESSAGE_ID`

The message identifier of the `javax.faces.context.Message`₀ to be created if the minimum length check fails. The message format string for this message may optionally include a {0} placeholder, which will be replaced by the configured minimum length.

Constructors

LengthValidator()

`public LengthValidator()`

Construct a Validator₀ with no preconfigured limits.

LengthValidator(int)

`public LengthValidator(int maximum)`

Construct a Validator₀ with the specified preconfigured limit.

Parameters:

`maximum` - Maximum value to allow

LengthValidator(int, int)

`public LengthValidator(int maximum, int minimum)`

Construct a Validator₀ with the specified preconfigured limits.

Parameters:

maximum - Maximum value to allow

minimum - Minimum value to allow

Throws:

java.lang.IllegalArgumentException - if a specified maximum value is less than a specified minimum value

Methods

doubleValue(Object)

```
protected double doubleValue(java.lang.Object attributeValue)  
    throws NumberFormatException
```

Return the specified attribute value, converted to a double.

Parameters:

attributeValue - The attribute value to be converted

Throws:

java.lang.NumberFormatException - if conversion is not possible

getMaximum()

```
public int getMaximum()
```

Return the maximum length to be enforced by this Validator₀, if isMaximumSet() returns true.

getMessage(FacesContext, String)

```
protected javax.faces.context.Message0  
    getMessage(javax.faces.context.FacesContext0 context,  
        java.lang.String messageId)
```

Return a javax.faces.context.Message₀ for the specified parameters.

Parameters:

context - The javax.faces.context.FacesContext₀ associated with the request being processed

messageId - Message identifier of the requested message

getMessage(FacesContext, String, Object[])

```
protected javax.faces.context.Message0
    getMessage(javax.faces.context.FacesContext0 context,
               java.lang.String messageId, java.lang.Object[] params)
```

Return a javax.faces.context.Message₀ for the specified parameters.

Parameters:

context - The javax.faces.context.FacesContext₀ associated with the request being processed

messageId - Message identifier of the requested message

params - Substitution parameters for this message

getMessageResources()

```
protected javax.faces.context.MessageResources0 getMessageResources()
```

Return the javax.faces.context.MessageResources₀ instance for the message resources defined by the JavaServer Faces Specification.

getMinimum()

```
public int getMinimum()
```

Return the minimum length to be enforced by this Validator₀, if isMinimumSet() returns true.

isMaximumSet()

```
public boolean isMaximumSet()
```

Return a flag indicating whether a maximum length has been set.

isMinimumSet()

```
public boolean isMinimumSet()
```

Return a flag indicating whether a minimum limit has been set.

longValue(Object)

```
protected long longValue(java.lang.Object attributeValue)
    throws NumberFormatException
```

Return the specified attribute value, converted to a long.

Parameters:

attributeValue - The attribute value to be converted

Throws:

java.lang.NumberFormatException - if conversion is not possible

setMaximum(int)

```
public void setMaximum(int maximum)
```

Set the maximum length to be enforced by this Validator_o.

Parameters:

maximum - The new maximum value

Throws:

java.lang.IllegalArgumentException - if a specified maximum value is less than a specified minimum value

setMinimum(int)

```
public void setMinimum(int minimum)
```

Set the minimum length to be enforced by this Validator_o.

Parameters:

minimum - The new minimum value

Throws:

java.lang.IllegalArgumentException - if a specified maximum value is less than a specified minimum value

stringValue(Object)

```
protected java.lang.String stringValue(java.lang.Object attributeValue)
```

Return the specified attribute value, converted to a String.

Parameters:

attributeValue - The attribute value to be converted

validate(FacesContext, UIComponent)

```
public void validate(javax.faces.context.FacesContexto context,
                     javax.faces.component.UIComponento component)
```

Description copied from interface: javax.faces.validator.Validator_o

Perform the correctness checks implemented by this Validator_o against the specified javax.faces.component.UIComponent_o. If any violations are found:

- Add zero or more javax.faces.context.Message_os to the specified javax.faces.context.FacesContext_o, specifying this

`javax.faces.component.UIComponent` as associated with the message, describing the nature of the violation(s) encountered.

- Set the `valid` property on the specified `javax.faces.component.UIComponent` to `false`.

Overrides: `validate` in class `ValidatorBase`

javax.faces.validator LongRangeValidator

Declaration

```
public class LongRangeValidator extends ValidatorBase  
  
java.lang.Object  
|  
+--javax.faces.validator.ValidatorBase  
|  
+--javax.faces.validator.LongRangeValidator
```

All Implemented Interfaces: `java.io.Serializable, Validator`

Description

LongRangeValidator is a `Validator`, that checks the value of the corresponding component against specified minimum and maximum values. The following algorithm is implemented:

- Call `getValue()` to retrieve the current value of the component. If it is `null`, exit immediately. (If `null` values should not be allowed, a `RequiredValidator` can be configured to check for this case.)
- If the current component value is not a floating point type, or a String that is convertible to long, add a `TYPE_MESSAGE_ID` message to the `javax.faces.context.FacesContext` for this request, and skip subsequent checks.
- If a `maximum` property has been configured on this `Validator`, check the component value against this limit. If the component value is greater than the specified minimum, add a `MAXIMUM_MESSAGE_ID` message to the `javax.faces.context.FacesContext` for this request.
- If a `minimum` property has been configured on this `Validator`, check the component value against this limit. If the component value is less than the specified minimum, add a `MINIMUM_MESSAGE_ID` message to the `javax.faces.context.FacesContext` for this request.

Member Summary
Fields

Member Summary

static	MAXIMUM_MESSAGE_ID ₀	
java.lang.String	The message identifier of the javax.faces.context.Message ₀ to be created if the maximum value check fails.	
static	MINIMUM_MESSAGE_ID ₀	
java.lang.String	The message identifier of the javax.faces.context.Message ₀ to be created if the minimum value check fails.	
static	TYPE_MESSAGE_ID ₀	
java.lang.String	The message identifier of the javax.faces.context.Message ₀ to be created if the current value of this component is not of the correct type.	

Constructors

LongRangeValidator()	
	Construct a Validator ₀ with no preconfigured limits.
LongRangeValidator(long maximum)	
	Construct a Validator ₀ with the specified preconfigured limit.
LongRangeValidator(long maximum, long minimum)	
	Construct a Validator ₀ with the specified preconfigured limits.

Methods

protected double	doubleValue(java.lang.Object attributeValue)	
	Return the specified attribute value, converted to a double.	
long	getMaximum()	
	Return the maximum value to be enforced by this Validator ₀ , if isMaximumSet() returns true.	
protected javax.faces.context.Message	getMessage(javax.faces.context.FacesContext context, java.lang.String messageId)	
	Return a javax.faces.context.Message ₀ for the specified parameters.	
protected javax.faces.context.Message	getMessage(javax.faces.context.FacesContext context, java.lang.String messageId, java.lang.Object params)	
	Return a javax.faces.context.Message ₀ for the specified parameters.	
protected javax.faces.context.MessageResources	getMessageResources()	
	Return the javax.faces.context.MessageResources ₀ instance for the message resources defined by the JavaServer Faces Specification.	
long	getMinimum()	
	Return the minimum value to be enforced by this Validator ₀ , if isMinimumSet() returns true.	
boolean	isMaximumSet()	
	Return a flag indicating whether a maximum limit has been set.	
boolean	isMinimumSet()	
	Return a flag indicating whether a minimum limit has been set.	
protected long	longValue(java.lang.Object attributeValue)	
	Return the specified attribute value, converted to a long.	
void	setMaximum(long maximum)	
	Set the maximum value to be enforced by this Validator ₀ .	
void	setMinimum(long minimum)	
	Set the minimum value to be enforced by this Validator ₀ .	

Member Summary

```
protected stringValue(java.lang.Object attributeValue)
java.lang.String      Return the specified attribute value, converted to a String.
void validate(javax.faces.context.FacesContext context,
               javax.faces.component.UIComponent component)_o
```

Inherited Member Summary

Methods inherited from class `Object`

```
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(),
toString(), wait(), wait(), wait()
```

Fields

`MAXIMUM_MESSAGE_ID`

```
public static final java.lang.String MAXIMUM_MESSAGE_ID
```

The message identifier of the `javax.faces.context.Message` to be created if the maximum value check fails. The message format string for this message may optionally include a { 0 } placeholder, which will be replaced by the configured maximum value.

`MINIMUM_MESSAGE_ID`

```
public static final java.lang.String MINIMUM_MESSAGE_ID
```

The message identifier of the `javax.faces.context.Message` to be created if the minimum value check fails. The message format string for this message may optionally include a { 0 } placeholder, which will be replaced by the configured minimum value.

`TYPE_MESSAGE_ID`

```
public static final java.lang.String TYPE_MESSAGE_ID
```

The message identifier of the `javax.faces.context.Message` to be created if the current value of this component is not of the correct type.

Constructors

`LongRangeValidator()`

```
public LongRangeValidator()
```

Construct a Validator₀ with no preconfigured limits.

LongRangeValidator(long)

```
public LongRangeValidator(long maximum)
```

Construct a Validator₀ with the specified preconfigured limit.

Parameters:

maximum - Maximum value to allow

LongRangeValidator(long, long)

```
public LongRangeValidator(long maximum, long minimum)
```

Construct a Validator₀ with the specified preconfigured limits.

Parameters:

maximum - Maximum value to allow

minimum - Minimum value to allow

Throws:

java.lang.IllegalArgumentException - if a specified maximum value is less than a specified minimum value

Methods

doubleValue(Object)

```
protected double doubleValue(java.lang.Object attributeValue)
    throws NumberFormatException
```

Return the specified attribute value, converted to a double.

Parameters:

attributeValue - The attribute value to be converted

Throws:

java.lang.NumberFormatException - if conversion is not possible

getMaximum()

```
public long getMaximum()
```

Return the maximum value to be enforced by this Validator₀, if isMaximumSet() returns true.

getMessage(FacesContext, String)

```
protected javax.faces.context.Message0
    getMessage(javax.faces.context.FacesContext0 context,
        java.lang.String messageId)
```

Return a javax.faces.context.Message₀ for the specified parameters.

Parameters:

context - The javax.faces.context.FacesContext₀ associated with the request being processed

messageId - Message identifier of the requested message

getMessage(FacesContext, String, Object[])

```
protected javax.faces.context.Message0
    getMessage(javax.faces.context.FacesContext0 context,
        java.lang.String messageId, java.lang.Object[] params)
```

Return a javax.faces.context.Message₀ for the specified parameters.

Parameters:

context - The javax.faces.context.FacesContext₀ associated with the request being processed

messageId - Message identifier of the requested message

params - Substitution parameters for this message

getMessageResources()

```
protected javax.faces.context.MessageResources0 getMessageResources()
```

Return the javax.faces.context.MessageResources₀ instance for the message resources defined by the JavaServer Faces Specification.

getMinimum()

```
public long getMinimum()
```

Return the minimum value to be enforced by this Validator₀, if isMinimumSet() returns true.

isMaximumSet()

```
public boolean isMaximumSet()
```

Return a flag indicating whether a maximum limit has been set.

isMinimumSet()

```
public boolean isMinimumSet()
```

Return a flag indicating whether a minimum limit has been set.

longValue(Object)

```
protected long longValue(java.lang.Object attributeValue)  
    throws NumberFormatException
```

Return the specified attribute value, converted to a long.

Parameters:

attributeValue - The attribute value to be converted

Throws:

java.lang.NumberFormatException - if conversion is not possible

setMaximum(long)

```
public void setMaximum(long maximum)
```

Set the maximum value to be enforced by this Validator_o.

Parameters:

maximum - The new maximum value

Throws:

java.lang.IllegalArgumentException - if a specified maximum value is less than a specified minimum value

setMinimum(long)

```
public void setMinimum(long minimum)
```

Set the minimum value to be enforced by this Validator_o.

Parameters:

minimum - The new minimum value

Throws:

java.lang.IllegalArgumentException - if a specified maximum value is less than a specified minimum value

stringValue(Object)

```
protected java.lang.String stringValue(java.lang.Object attributeValue)
```

Return the specified attribute value, converted to a String.

Parameters:

attributeValue - The attribute value to be converted

validate(FacesContext, UIComponent)

```
public void validate(javax.faces.context.FacesContext0 context,  
                     javax.faces.component.UIComponent0 component)
```

Description copied from interface: javax.faces.validator.Validator₀

Perform the correctness checks implemented by this Validator₀ against the specified javax.faces.component.UIComponent₀. If any violations are found:

- Add zero or more javax.faces.context.Message₀s to the specified javax.faces.context.FacesContext₀, specifying this javax.faces.component.UIComponent₀ as associated with the message, describing the nature of the violation(s) encountered.
- Set the valid property on the specified javax.faces.component.UIComponent₀ to false.

Overrides: validate in class ValidatorBase

javax.faces.validator RequiredValidator

Declaration

```
public class RequiredValidator extends ValidatorBase

java.lang.Object
|
+--javax.faces.validator.ValidatorBase
|
+--javax.faces.validator.RequiredValidator
```

All Implemented Interfaces: `java.io.Serializable, Validator`

Description

RequiredValidator is a `Validator`, that checks for the existence of a value for the associated component. For the purposes of this Validator, existence means a non-null value returned by calling `getValue()`. In addition, if the component value is a String, it must have a length greater than zero.

Member Summary

Fields

static	FAILED_MESSAGE_ID ₀
java.lang.String	The message identifier of the <code>javax.faces.context.Message</code> to be created if this validation fails.

Constructors

`RequiredValidator()`

Methods

protected double	doubleValue(java.lang.Object attributeValue)	Return the specified attribute value, converted to a double.
protected javax.faces.context.Message	getMessage(javax.faces.context.FacesContext context, java.lang.String messageId)	Return a <code>javax.faces.context.Message</code> for the specified parameters.
protected javax.faces.context.Message	getMessage(javax.faces.context.FacesContext context, java.lang.String messageId, java.lang.Object params)	Return a <code>javax.faces.context.Message</code> for the specified parameters.
protected javax.faces.context.MessageResources	getMessageResources()	Return the <code>javax.faces.context.MessageResources</code> instance for the message resources defined by the JavaServer Faces Specification.

Member Summary

```
protected long longValue(java.lang.Object attributeValue)
    Return the specified attribute value, converted to a long.
protected stringValue(java.lang.Object attributeValue)
java.lang.String      Return the specified attribute value, converted to a String.
void validate(javax.faces.context.FacesContext context,
              javax.faces.component.UIComponent component)_o
```

Inherited Member Summary**Methods inherited from class Object**

```
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(),
toString(), wait(), wait(), wait()
```

Fields

FAILED_MESSAGE_ID

```
public static final java.lang.String FAILED_MESSAGE_ID
```

The message identifier of the `javax.faces.context.Message`_o to be created if this validation fails.

Constructors

RequiredValidator()

```
public RequiredValidator()
```

Methods

doubleValue(Object)

```
protected double doubleValue(java.lang.Object attributeValue)
    throws NumberFormatException
```

Return the specified attribute value, converted to a double.

Parameters:

`attributeValue` - The attribute value to be converted

Throws:

java.lang.NumberFormatException - if conversion is not possible

getMessage(FacesContext, String)

```
protected javax.faces.context.Message0
    getMessage(javax.faces.context.FacesContext0 context,
    java.lang.String messageId)
```

Return a javax.faces.context.Message₀ for the specified parameters.

Parameters:

context - The javax.faces.context.FacesContext₀ associated with the request being processed
messageId - Message identifier of the requested message

getMessage(FacesContext, String, Object[])

```
protected javax.faces.context.Message0
    getMessage(javax.faces.context.FacesContext0 context,
    java.lang.String messageId, java.lang.Object[] params)
```

Return a javax.faces.context.Message₀ for the specified parameters.

Parameters:

context - The javax.faces.context.FacesContext₀ associated with the request being processed
messageId - Message identifier of the requested message
params - Substitution parameters for this message

getMessageResources()

```
protected javax.faces.context.MessageResources0 getMessageResources( )
```

Return the javax.faces.context.MessageResources₀ instance for the message resources defined by the JavaServer Faces Specification.

longValue(Object)

```
protected long longValue(java.lang.Object attributeValue)
    throws NumberFormatException
```

Return the specified attribute value, converted to a long.

Parameters:

attributeValue - The attribute value to be converted

Throws:

java.lang.NumberFormatException - if conversion is not possible

stringValue(Object)

```
protected java.lang.String stringValue(java.lang.Object attributeValue)
```

Return the specified attribute value, converted to a String.

Parameters:

attributeValue - The attribute value to be converted

validate(FacesContext, UIComponent)

```
public void validate(javax.faces.context.FacesContext0 context,  
                     javax.faces.component.UIComponent0 component)
```

Description copied from interface: javax.faces.validator.Validator₀

Perform the correctness checks implemented by this Validator₀ against the specified javax.faces.component.UIComponent₀. If any violations are found:

- Add zero or more javax.faces.context.Message₀s to the specified javax.faces.context.FacesContext₀, specifying this javax.faces.component.UIComponent₀ as associated with the message, describing the nature of the violation(s) encountered.
- Set the valid property on the specified javax.faces.component.UIComponent₀ to false.

Overrides: validate in class ValidatorBase

javax.faces.validator StringRangeValidator

Declaration

```
public class StringRangeValidator extends ValidatorBase  
  
java.lang.Object  
|  
+--javax.faces.validator.ValidatorBase  
|  
+--javax.faces.validator.StringRangeValidator
```

All Implemented Interfaces: java.io.Serializable, Validator₀

Description

StringRangeValidator is a Validator₀ that checks the value of the corresponding component against specified minimum and maximum values. The following algorithm is implemented:

- Call `getValue()` to retrieve the current value of the component. If it is `null`, exit immediately. (If `null` values should not be allowed, a `RequiredValidator0` can be configured to check for this case.)
- Convert the current component value to `String`, if necessary, by calling `toString()`.
- If a `maximum` property has been configured on this Validator₀, check the component value against this limit. If the component value is greater than the specified minimum, add a `MAXIMUM_MESSAGE_ID` message to the `javax.faces.context.FacesContext0` for this request.
- If a `minimum` property has been configured on this Validator₀, check the component value against this limit. If the component value is less than the specified minimum, add a `MINIMUM_MESSAGE_ID` message to the `javax.faces.context.FacesContext0` for this request.

Member Summary

Fields

static MAXIMUM_MESSAGE_ID ₀	
java.lang.String	The message identifier of the <code>javax.faces.context.Message₀</code> to be created if the maximum value check fails.

Member Summary

static MINIMUM_MESSAGE_ID₀
java.lang.String The message identifier of the javax.faces.context.Message₀ to be created if the minimum value check fails.

Constructors

StringRangeValidator()₀
Construct a Validator₀ with no preconfigured limits.
StringRangeValidator(java.lang.String maximum)₀
Construct a Validator₀ with the specified preconfigured limit.
StringRangeValidator(java.lang.String maximum,
java.lang.String minimum)₀
Construct a Validator₀ with the specified preconfigured limits.

Methods

protected double doubleValue(java.lang.Object attributeValue)
Return the specified attribute value, converted to a double.
java.lang.String getMaximum()₀
Return the maximum value to be enforced by this Validator₀, if
isMaximumSet() returns true.
protected getMessage(javax.faces.context.FacesContext context,
javax.faces.context.Message₀ message)
Return a javax.faces.context.Message₀ for the specified parameters.
protected getMessage(javax.faces.context.FacesContext context,
javax.faces.context.Message₀ messageId, java.lang.Object params)
Return a javax.faces.context.Message₀ for the specified parameters.
protected getMessageResources()
Return the javax.faces.context.MessageResources₀ instance for the
message resources defined by the JavaServer Faces Specification.
java.lang.String getMinimum()₀
Return the minimum value to be enforced by this Validator₀, if
isMinimumSet() returns true.
boolean isMaximumSet()₀
Return a flag indicating whether a maximum limit has been set.
boolean isMinimumSet()₀
Return a flag indicating whether a minimum limit has been set.
protected long longValue(java.lang.Object attributeValue)
Return the specified attribute value, converted to a long.
void setMaximum(java.lang.String maximum)₀
Set the maximum value to be enforced by this Validator₀.
void setMinimum(java.lang.String minimum)₀
Set the minimum value to be enforced by this Validator₀.
protected stringValue(java.lang.Object attributeValue)
java.lang.String Return the specified attribute value, converted to a String.
void validate(javax.faces.context.FacesContext context,
javax.faces.component.UIComponent component)₀

Inherited Member Summary

Methods inherited from class Object

`clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(),
toString(), wait(), wait(), wait()`

Fields

MAXIMUM_MESSAGE_ID

`public static final java.lang.String MAXIMUM_MESSAGE_ID`

The message identifier of the `javax.faces.context.Message`₀ to be created if the maximum value check fails. The message format string for this message may optionally include a {0} placeholder, which will be replaced by the configured maximum value.

MINIMUM_MESSAGE_ID

`public static final java.lang.String MINIMUM_MESSAGE_ID`

The message identifier of the `javax.faces.context.Message`₀ to be created if the minimum value check fails. The message format string for this message may optionally include a {0} placeholder, which will be replaced by the configured minimum value.

Constructors

StringRangeValidator()

`public StringRangeValidator()`

Construct a Validator₀ with no preconfigured limits.

StringRangeValidator(String)

`public StringRangeValidator(java.lang.String maximum)`

Construct a Validator₀ with the specified preconfigured limit.

Parameters:

`maximum` - Maximum value to allow (if any)

Throws:

`NullPointerException` - if a specified limit is null

StringRangeValidator(String, String)

```
public StringRangeValidator(java.lang.String maximum, java.lang.String minimum)
```

Construct a Validator_o with the specified preconfigured limits.

Parameters:

maximum - Maximum value to allow

minimum - Minimum value to allow

Throws:

java.lang.IllegalArgumentException - if a specified maximum value is less than a specified minimum value

NullPointerException - if a specified limit is null

Methods

doubleValue(Object)

```
protected double doubleValue(java.lang.Object attributeValue)
    throws NumberFormatException
```

Return the specified attribute value, converted to a double.

Parameters:

attributeValue - The attribute value to be converted

Throws:

java.lang.NumberFormatException - if conversion is not possible

getMaximum()

```
public java.lang.String getMaximum()
```

Return the maximum value to be enforced by this Validator_o, if isMaximumSet() returns true.

getMessage(FacesContext, String)

```
protected javax.faces.context.Messageo
    getMessage(javax.faces.context.FacesContexto context,
        java.lang.String messageId)
```

Return a javax.faces.context.Message_o for the specified parameters.

Parameters:

context - The javax.faces.context.FacesContext_o associated with the request being processed

`messageId` - Message identifier of the requested message

getMessage(FacesContext, String, Object[])

```
protected javax.faces.context.Message0
    getMessage(javax.faces.context.FacesContext0 context,
               java.lang.String messageId, java.lang.Object[] params)
```

Return a `javax.faces.context.Message0` for the specified parameters.

Parameters:

`context` - The `javax.faces.context.FacesContext0` associated with the request being processed

`messageId` - Message identifier of the requested message

`params` - Substitution parameters for this message

getMessageResources()

```
protected javax.faces.context.MessageResources0 getMessageResources()
```

Return the `javax.faces.context.MessageResources0` instance for the message resources defined by the JavaServer Faces Specification.

getMinimum()

```
public java.lang.String getMinimum()
```

Return the minimum value to be enforced by this `Validator0`, if `isMinimumSet()` returns true.

isMaximumSet()

```
public boolean isMaximumSet()
```

Return a flag indicating whether a maximum limit has been set.

isMinimumSet()

```
public boolean isMinimumSet()
```

Return a flag indicating whether a minimum limit has been set.

longValue(Object)

```
protected long longValue(java.lang.Object attributeValue)
    throws NumberFormatException
```

Return the specified attribute value, converted to a `long`.

Parameters:

attributeValue - The attribute value to be converted

Throws:

java.lang.NumberFormatException - if conversion is not possible

setMaximum(String)

```
public void setMaximum(java.lang.String maximum)
```

Set the maximum value to be enforced by this Validator_o.

Parameters:

maximum - The new maximum value

Throws:

java.lang.IllegalArgumentException - if a specified maximum value is less than a specified minimum value

NullPointerException - if maximum is null

setMinimum(String)

```
public void setMinimum(java.lang.String minimum)
```

Set the minimum value to be enforced by this Validator_o.

Parameters:

minimum - The new minimum value

Throws:

java.lang.IllegalArgumentException - if a specified maximum value is less than a specified minimum value

NullPointerException - if minimum is null

stringValue(Object)

```
protected java.lang.String stringValue(java.lang.Object attributeValue)
```

Return the specified attribute value, converted to a String.

Parameters:

attributeValue - The attribute value to be converted

validate(FacesContext, UIComponent)

```
public void validate(javax.faces.context.FacesContexto context,
                     javax.faces.component.UIComponento component)
```

Description copied from interface: javax.faces.validator.Validator₀

Perform the correctness checks implemented by this Validator₀ against the specified javax.faces.component.UIComponent₀. If any violations are found:

- Add zero or more javax.faces.context.Message₀s to the specified javax.faces.context.FacesContext₀, specifying this javax.faces.component.UIComponent₀ as associated with the message, describing the nature of the violation(s) encountered.
- Set the valid property on the specified javax.faces.component.UIComponent₀ to false.

Overrides: validate in class ValidatorBase

javax.faces.validator Validator

Declaration

```
public interface Validator extends java.io.Serializable
```

All Superinterfaces: `java.io.Serializable`

All Known Implementing Classes: `ValidatorBase`

Description

A **Validator** implementation is a class that can perform validation (correctness checks) on a `javax.faces.component.UIComponent0`. Zero or more Validators can be associated with each `javax.faces.component.UIComponent0` in the component tree, and are called during the *Process Validations* phase of the request processing lifecycle.

Individual `Validator0`s should examine the component that they are passed, and add `javax.faces.context.Message0` instances to the `javax.faces.context.FacesContext0` for the current request, documenting any failures to conform to the required rules. In general, such messages should be associated with the `javax.faces.component.UIComponent0` on which the validation failure occurred. In addition, the `valid` property of the corresponding `javax.faces.component.UIComponent0` should be set to `false` on validation failures.

For maximum generality, `Validator0` instances may be configurable based on properties of the `Validator0` implementation class. For example, a range check `Validator0` might support configuration of the minimum and maximum values to be used. In addition, because `Validator0`s are part of the saved and restored state of a component tree, classes that implement this interface must also be serializable.

Member Summary

Methods

```
void validate(javax.faces.context.FacesContext context,  
             javax.faces.component.UIComponent component)0  
    Perform the correctness checks implemented by this Validator0 against the  
    specified javax.faces.component.UIComponent0.
```

Methods

validate(FacesContext, UIComponent)

```
public void validate(javax.faces.context.FacesContext0 context,  
                     javax.faces.component.UIComponent0 component)
```

Perform the correctness checks implemented by this Validator₀ against the specified javax.faces.component.UIComponent₀. If any violations are found:

- Add zero or more javax.faces.context.Message₀s to the specified javax.faces.context.FacesContext₀, specifying this javax.faces.component.UIComponent₀ as associated with the message, describing the nature of the violation(s) encountered.
- Set the valid property on the specified javax.faces.component.UIComponent₀ to false.

Parameters:

context - FacesContext for the request we are processing

component - UIComponent we are checking for correctness

Throws:

NullPointerException - if context or component is null

JSF.10.10 Package `javax.faces.webapp`

Package `javax.faces.webapp`

Description

Classes required for integration of JavaServer Faces int web applications, including a standard servlet, base classes for JSP custom component tags, and concrete tag implementations for core tags.

Class Summary

Classes

<code>AttributeTag₀</code>	Tag implementation that adds an attribute with a specified name and String value to the component whose tag it is nested inside, if the component does not already contain an attribute with the same name.
<code>FacesBodyTag₀</code>	FacesBodyTag is a base class for all JSP custom actions that need to process their tag bodies.
<code>FacesServlet₀</code>	FacesServlet is a servlet that manages the request processing lifecycle for web applications that are utilizing JavaServer Faces to construct the user interface.
<code>FacesTag₀</code>	FacesTag is a base class for all JSP custom actions that correspond to user interface components in a page that is rendered by JavaServer Faces.
<code>FacetTag₀</code>	FacetTag is the JSP mechanism for denoting a <code>javax.faces.component.UIComponent₀</code> to be added as a facet to the component associated with its parent.
<code>JspResponseWriter₀</code>	JspResponseWriter is a concrete implementation of <code>javax.faces.context.ResponseWriter₀</code> that wraps a <code>PageContext</code> passed to its constructor.
<code>ServletResponseWriter₀</code>	ServletResponseWriter is a concrete implementation of <code>javax.faces.context.ResponseWriter₀</code> that wraps a <code>PrintWriter</code> passed to its constructor.
<code>ValidatorTag₀</code>	ValidatorTag is a base class for all JSP custom actions that create and register a <code>Validator</code> instance on the <code>javax.faces.component.UIComponent₀</code> associated with our most immediate surrounding instance of a tag whose implementation class is a subclass of <code>FacesTag₀</code> .

javax.faces.webapp AttributeTag

Declaration

```
public class AttributeTag extends javax.servlet.jsp.tagext.TagSupport  
  
java.lang.Object  
|  
+--javax.servlet.jsp.tagext.TagSupport  
|  
+--javax.faces.webapp.AttributeTag
```

All Implemented Interfaces: javax.servlet.jsp.tagext.IterationTag,
java.io.Serializable, javax.servlet.jsp.tagext.Tag

Description

Tag implementation that adds an attribute with a specified name and String value to the component whose tag it is nested inside, if the component does not already contain an attribute with the same name. This tag creates no output to the page currently being created.

Member Summary

Constructors

```
AttributeTag()
```

Methods

```
int doStartTag()  
    Register the specified attribute name and value with the  
    javax.faces.component.UIComponent0 instance associated with our most  
    immediately surrounding FacesTag0 instance, if this  
    javax.faces.component.UIComponent0 does not already have a value for  
    the specified attribute name.  
void release()  
    Release references to any acquired resources.  
void setName(java.lang.String name)  
    Set the attribute name.  
void setValue(java.lang.String value)  
    Set the attribute value.
```

Inherited Member Summary

Fields inherited from interface `IterationTag`

`EVAL_BODY AGAIN`

Fields inherited from interface `Tag`

`EVAL_BODY_INCLUDE, EVAL_PAGE, SKIP_BODY, SKIP_PAGE`

Fields inherited from class `TagSupport`

`id, pageContext`

Methods inherited from class `Object`

`clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(),
toString(), wait(), wait(), wait()`

Methods inherited from class `TagSupport`

`doAfterBody(), doEndTag(), findAncestorWithClass(Tag, Class), getId(), getParent(),
getValue(String), getValues(), removeValue(String), setId(String),
setPageContext(PageContext), setParent(Tag), setValue(String, Object)`

Constructors

`AttributeTag()`

```
public AttributeTag()
```

Methods

`doStartTag()`

```
public int doStartTag()  
throws JspException
```

Register the specified attribute name and value with the

`javax.faces.component.UIComponent0` instance associated with our most immediately surrounding `FacesTag0` instance, if this `javax.faces.component.UIComponent0` does not already have a value for the specified attribute name.

Overrides: `doStartTag` in class `TagSupport`

Throws:

`javax.servlet.jsp.JspException` - if a JSP error occurs

release()

```
public void release()
```

Release references to any acquired resources.

Overrides: release in class TagSupport

setName(String)

```
public void setName(java.lang.String name)
```

Set the attribute name.

Parameters:

name - The new attribute name

setValue(String)

```
public void setValue(java.lang.String value)
```

Set the attribute value.

Parameters:

value - The new attribute value

javax.faces.webapp FacesBodyTag

Declaration

```
public abstract class FacesBodyTag extends FacesTag_ implements  
    javax.servlet.jsp.tagext.BodyTag  
  
java.lang.Object  
|  
+--javax.servlet.jsp.tagext.TagSupport  
|  
+--javax.faces.webapp.FacesTag_  
|  
+--javax.faces.webapp.FacesBodyTag
```

All Implemented Interfaces: javax.servlet.jsp.tagext.BodyTag,
javax.servlet.jsp.tagext.IterationTag, java.io.Serializable,
javax.servlet.jsp.tagext.Tag

Description

FacesBodyTag is a base class for all JSP custom actions that need to process their tag bodies.

Member Summary

Fields

```
protected bodyContent_  
javax.servlet.jsp.tage      The bodyContent for this tag handler.  
xt.BodyContent
```

Constructors

```
FacesBodyTag()
```

Methods

```
void doInitBody()  
        Prepare for evaluation of the body.  
getBodyContent()  
javax.servlet.jsp.tage      Return the BodyContent for this tag handler.  
xt.BodyContent  
protected int getDoStartValue()  
getPreviousOut()  
javax.servlet.jsp.JspW      Get the JspWriter from our BodyContent.  
riter  
void release()
```

Member Summary

```
void setBodyContent(javax.servlet.jsp.tagext.BodyContent  
bodyContent)0  
Set the bodyContent for this tag handler.
```

Inherited Member Summary**Fields inherited from interface BodyTag**

EVAL_BODY_BUFFERED, EVAL_BODY_TAG

Fields inherited from class FacesTag₀

COMPONENT_STACK_ATTR₀, childIndex₀, component₀, componentStack₀, context₀, created₀, modelReference₀, numChildren₀, rendered₀

Fields inherited from interface IterationTag

EVAL_BODY AGAIN

Fields inherited from interface Tag

EVAL_BODY_INCLUDE, EVAL_PAGE, SKIP_BODY, SKIP_PAGE

Fields inherited from class TagSupport

id, pageContext

Methods inherited from class FacesTag₀

createComponent()0, doAfterBody()0, doEndTag()0, doStartTag()0, encodeBegin()0, encodeChildren()0, encodeEnd()0, findComponent()0, findComponentStack()0, getChildIndex()0, getComponent()0, getCreated()0, getDoAfterBodyValue()0, getDoEndValue()0, getModelReference()0, getNearestEnclosingFacesTag()0, getNumChildren()0, getRendered()0, getRendererType()0, incrementNumChildren()0, isSuppressed()0, overrideProperties(UIComponent)0, setModelReference(String)0, setParent(Tag)0, setRendered(boolean)0, setupResponseWriter()0

Methods inherited from interface IterationTag

doAfterBody()

Methods inherited from class Object

clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(), wait(), wait()

Methods inherited from interface Tag

doEndTag(), doStartTag(), getParent(), setPageContext(PageContext), setParent(Tag)

Inherited Member Summary

Methods inherited from class `TagSupport`

```
findAncestorWithClass(Tag, Class), getId(), getParent(), getValue(String),
getValues(), removeValue(String), setId(String), setPageContext(PageContext),
setValue(String, Object)
```

Fields

`bodyContent`

```
protected javax.servlet.jsp.tagext.BodyContent bodyContent
```

The `bodyContent` for this tag handler.

Constructors

`FacesBodyTag()`

```
public FacesBodyTag()
```

Methods

`doInitBody()`

```
public void doInitBody()
    throws JspException
```

Prepare for evaluation of the body. This method is invoked by the JSP page implementation object after `setBodyContent()` and before the first time the body is to be evaluated. This method will not be invoked for empty tags or for non-empty tags whose `doStartTag()` method returns `SKIP_BODY` or `EVAL_BODY_INCLUDE`.

Specified By: `doInitBody` in interface `BodyTag`

Throws:

`javax.servlet.jsp.JspException` - if an error is encountered

`getBodyContent()`

```
public javax.servlet.jsp.tagext.BodyContent getBodyContent()
```

Return the `BodyContent` for this tag handler.

getDoStartValue()

```
protected int getDoStartValue()
    throws JspException
```

Description copied from class: `javax.faces.webapp.FacesTag`

Return the flag value that should be returned from the `doStart()` method when it is called. Subclasses may override this method to return the appropriate value.

Overrides: `getDoStartValue` in class `FacesTag`

Throws:

```
javax.servlet.jsp.JspException
```

getPreviousOut()

```
public javax.servlet.jsp.JspWriter getPreviousOut()
```

Get the `JspWriter` from our `BodyContent`.

release()

```
public void release()
```

Description copied from class: `javax.faces.webapp.FacesTag`

Release any resources allocated during the execution of this tag handler.

Specified By: `release` in interface `Tag`

Overrides: `release` in class `FacesTag`

setBodyContent(BodyContent)

```
public void setBodyContent(javax.servlet.jsp.tagext.BodyContent bodyContent)
```

Set the `bodyContent` for this tag handler. This method is invoked by the JSP page implementation object at most once per action invocation, before `doInitiBody()`. This method will not be invoked for empty tags or for non-empty tags whose `doStartTag()` method returns `SKIP_BODY` or `EVAL_BODY_INCLUDE`.

Specified By: `setBodyContent` in interface `BodyTag`

Parameters:

`bodyContent` - The new `BodyContent` for this tag

javax.faces.webapp FacesServlet

Declaration

```
public final class FacesServlet implements javax.servlet.Servlet  
  
java.lang.Object  
|  
+--javax.faces.webapp.FacesServlet
```

All Implemented Interfaces: javax.servlet.Servlet

Description

FacesServlet is a servlet that manages the request processing lifecycle for web applications that are utilizing JavaServer Faces to construct the user interface.

This servlet recognizes the following context initialization parameters:

- **javax.faces.lifecycle.LIFECYCLE_ID** - Lifecycle identifier of the `javax.faces.lifecycle.Lifecycle` instance to be used when processing JSF requests in this web application. If not specified, the default instance, identified by `LifecycleFactory.DEFAULT_LIFECYCLE`, will be used.

Member Summary

Constructors

```
FacesServlet()
```

Methods

<code>void destroy()</code>	Release all resources acquired at startup time.
<code>getServletConfig()</code>	Return the <code>ServletConfig</code> instance for this servlet.
<code>javax.servlet.ServletConfig</code>	
<code>java.lang.String getServletInfo()</code>	Return information about this Servlet.
<code>void init(javax.servlet.ServletConfig servletConfig)</code>	Acquire the factory instances we will require.
<code>void service(javax.servlet.ServletRequest request,</code>	
<code> javax.servlet.ServletResponse response)</code>	Process an incoming request, and create the corresponding response, by executing the request processing lifecycle.

Inherited Member Summary

Methods inherited from class Object

`clone()`, `equals(Object)`, `finalize()`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`,
`toString()`, `wait()`, `wait()`, `wait()`

Constructors

FacesServlet()

```
public FacesServlet()
```

Methods

destroy()

```
public void destroy()
```

Release all resources acquired at startup time.

Specified By: `destroy` in interface `Servlet`

getServletConfig()

```
public javax.servlet.ServletConfig getServletConfig()
```

Return the `ServletConfig` instance for this servlet.

Specified By: `getServletConfig` in interface `Servlet`

getServletInfo()

```
public java.lang.String getServletInfo()
```

Return information about this Servlet.

Specified By: `getServletInfo` in interface `Servlet`

init(ServletConfig)

```
public void init(javax.servlet.ServletConfig servletConfig)
                 throws ServletException
```

Acquire the factory instances we will require.

Specified By: init in interface Servlet

Throws:

javax.servlet.ServletException

service(ServletRequest, ServletResponse)

```
public void service(javax.servlet.ServletRequest request,
                    javax.servlet.ServletResponse response)
                    throws IOException, ServletException
```

Process an incoming request, and create the corresponding response, by executing the request processing lifecycle.

Specified By: service in interface Servlet

Parameters:

request - The servlet request we are processing

response - The servlet response we are creating

Throws:

java.io.IOException - if an input/output error occurs during processing

javax.servlet.ServletException - if a servlet error occurs during processing

javax.faces.webapp FacesTag

Declaration

```
public abstract class FacesTag extends javax.servlet.jsp.tagext.TagSupport  
  
java.lang.Object  
|  
+--javax.servlet.jsp.tagext.TagSupport  
|  
+--javax.faces.webapp.FacesTag
```

All Implemented Interfaces: javax.servlet.jsp.tagext.IterationTag,
java.io.Serializable, javax.servlet.jsp.tagext.Tag

Direct Known Subclasses: FacesBodyTag₀

Description

FacesTag is a base class for all JSP custom actions that correspond to user interface components in a page that is rendered by JavaServer Faces. Tags that need to process their tag bodies should subclass FacesBodyTag₀ instead.

The **id** attribute of a **FacesTag** is used to set the `componentId` property of the `javax.faces.component.UIComponent0` associated with this tag.

Member Summary

Fields

protected int	childIndex ₀	This ivar is the index of this child Tag in its parent's child list.
protected javax.faces.component.UIComponent	component ₀	The <code>javax.faces.component.UIComponent₀</code> that is being encoded by this tag, if any.
protected static java.lang.String	COMPONENT_STACK_ATTR ₀	The request attribute under which our component stack is stored.
protected java.util.Stack	componentStack ₀	The <code>javax.faces.component.UIComponent₀</code> stack representing the current nesting of components for the current response.
protected javax.faces.context.FacesContext	context ₀	The <code>javax.faces.context.FacesContext₀</code> for the request being processed, if any.

Member Summary

```
protected boolean created0
Was a new component instance dynamically created when our findComponent() method was called.

protected modelReference0
java.lang.String An override for the model reference expression associated with our javax.faces.component.UIComponent0, if not null.

protected int numChildren0
This ivar is the number of children we have.

protected boolean rendered0
An override for the rendered attribute associated with our javax.faces.component.UIComponent0, if not true.
```

Constructors

```
FacesTag( )0
```

Methods

```
abstract createComponent()0
javax.faces.component.UIComponent Create and return a new javax.faces.component.UIComponent0 that is acceptable to this tag.

int doAfterBody()0
Handle the ending of the nested body content for a tag that implements javax.servlet.jsp.tagext.IterationTag, as all subclasses of this class do.

int doEndTag()0
Render the ending of the javax.faces.component.UIComponent0 that is associated with this tag (via the id attribute), by following these steps.

int doStartTag()0
Render the beginning of the javax.faces.component.UIComponent0 that is associated with this tag (via the id attribute), by following these steps.

protected void encodeBegin()0
Delegate to the encodeBegin() method of our corresponding javax.faces.component.UIComponent0.

protected void encodeChildren()0
Delegate to the encodeChildren() method of our corresponding javax.faces.component.UIComponent0.

protected void encodeEnd()0
Delegate to the encodeEnd() method of our corresponding javax.faces.component.UIComponent0.

protected findComponent()0
javax.faces.component.UIComponent Find and return the component, from the component tree, that corresponds to the relative identifier defined by the id attribute of this tag.

protected findComponentStack()0
java.util.Stack Locate and return the component stack for this response, creating one if this has not been done already.

protected int getChildIndex()0
Returns current value of childIndex.
```

Member Summary

javax.faces.component. UIComponent	getComponent() Return the javax.faces.component.UIComponent ₀ instance that is associated with this tag instance.
boolean	getCreated() Return true if we dynamically created a new component instance during our doStartTag() method.
protected int	getDoAfterBodyValue() Return the flag value that should be returned from the doAfterBody() method when it is called.
protected int	getDoEndValue() Return the flag value that should be returned from the doEnd() method when it is called.
protected int	getDoStartValue() Return the flag value that should be returned from the doStart() method when it is called.
java.lang.String	getModelReference() Return the override for the model reference expression.
protected javax.servlet.jsp.tag ext.Tag	getNearestEnclosingFacesTag() Returns the nearest enclosing FacesTag.
protected int	getNumChildren() Returns current value of numChildren.
boolean	getRendered() Return the override for the rendered attribute.
abstract java.lang.String	getRendererType() Return the rendererType property that selects the Renderer to be used for encoding this component, or null to ask the component to render itself directly.
protected void	incrementNumChildren() Increments current value of numChildren.
protected boolean	isSuppressed() Return true if rendering should be suppressed because some parent component on the stack has been configured with getRendersChildren() as true.
protected void	overrideProperties(javax.faces.component.UIComponent component) Override properties of the specified component if the corresponding properties of this tag handler were explicitly set, and the corresponding attribute of the component is not set.
void	release() Release any resources allocated during the execution of this tag handler.
void	setModelReference(java.lang.String modelReference) Set an override for the model reference expression.
void	setParent(javax.servlet.jsp.tagext.Tag t) Override this method to do three things.
void	setRendered(boolean rendered) Set an override for the rendered attribute.

Member Summary

```
protected void setupResponseWriter()0
    Set up the javax.faces.context.ResponseWriter0 for the current
    response, if this has not been done already.
```

Inherited Member Summary**Fields inherited from interface IterationTag**

EVAL_BODY AGAIN

Fields inherited from interface Tag

EVAL_BODY_INCLUDE, EVAL_PAGE, SKIP_BODY, SKIP_PAGE

Fields inherited from class TagSupport

id, pageContext

Methods inherited from class Objectclone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(),
toString(), wait(), wait(), wait()**Methods inherited from class TagSupport**findAncestorWithClass(Tag, Class), getId(), getParent(), getValue(String),
getValues(), removeValue(String), setId(String), setPageContext(PageContext),
setValue(String, Object)

Fields

childIndex

```
protected int childIndex
```

This ivar is the index of this child Tag in its parent's child list.

component

```
protected javax.faces.component.UIComponent0 component
```

The javax.faces.component.UIComponent₀ that is being encoded by this tag, if any.

COMPONENT_STACK_ATTR

```
protected static final java.lang.String COMPONENT_STACK_ATTR
```

The request attribute under which our component stack is stored.

componentStack

```
protected java.util.Stack componentStack
```

The javax.faces.component.UIComponent₀ stack representing the current nesting of components for the current response.

context

```
protected javax.faces.context.FacesContext context
```

The javax.faces.context.FacesContext₀ for the request being processed, if any.

created

```
protected boolean created
```

Was a new component instance dynamically created when our findComponent() method was called.

modelReference

```
protected java.lang.String modelReference
```

An override for the model reference expression associated with our javax.faces.component.UIComponent₀, if not null.

numChildren

```
protected int numChildren
```

This ivar is the number of children we have.

rendered

```
protected boolean rendered
```

An override for the rendered attribute associated with our javax.faces.component.UIComponent₀, if not true.

Constructors

FacesTag()

```
public FacesTag()
```

Methods

createComponent()

```
public abstract javax.faces.component.UIComponent createComponent()
```

Create and return a new `javax.faces.component.UIComponent` that is acceptable to this tag. Concrete subclasses must override this method.

doAfterBody()

```
public int doAfterBody()
    throws JspException
```

Handle the ending of the nested body content for a tag that implements `javax.servlet.jsp.tagext.IterationTag`, as all subclasses of this class do. The default implementation simply calls `getDoAfterBodyValue()` to retrieve the flag value to be returned.

Overrides: `doAfterBody` in class `TagSupport`

Throws:

`javax.servlet.jsp.JspException` - if an error occurs

doEndTag()

```
public int doEndTag()
    throws JspException
```

Render the ending of the `javax.faces.component.UIComponent` that is associated with this tag (via the `id` attribute), by following these steps.

- If the `rendersChildren` property of this component is `true`, call the `encodeChildren()` method of the component.
- Call the `encodeEnd()` method of the component.
- Release all references to the component, and pop it from the component stack for this response, removing the stack if this was the outermost component.

The flag value to be returned is acquired by calling the `getDoEndValue()` method, which tag subclasses may override if they do not want the default value.

Overrides: doEndTag in class TagSupport

Throws:

javax.servlet.jsp.JspException - if an error occurs

doStartTag()

```
public int doStartTag()
    throws JspException
```

Render the beginning of the javax.faces.component.UIComponent₀ that is associated with this tag (via the id attribute), by following these steps.

- Ensure that an appropriate javax.faces.context.ResponseWriter₀ is associated with the current javax.faces.context.FacesContext₀. This ensures that encoded output from the components is routed through the JspWriter for the current page.
- Locate the component (in the component tree) corresponding to this tag, creating a new one if necessary.
- Override the attributes of the associated component with values set in our custom tag attributes, if values for the corresponding attributes are NOT already set on the component.
- Push this component onto the stack of components corresponding to nested component tags for the current response, creating the stack if necessary.
- Call the encodeBegin() method of the component.

The flag value to be returned is acquired by calling the getDoStartValue() method, which tag subclasses may override if they do not want the default value.

Overrides: doStartTag in class TagSupport

Throws:

javax.servlet.jsp.JspException - if an error occurs

encodeBegin()

```
protected void encodeBegin()
    throws IOException
```

Delegate to the encodeBegin() method of our corresponding javax.faces.component.UIComponent₀. This method is called from doStartTag(). Normally, delegation occurs unconditionally; however, this method is abstracted out so that advanced tags can conditionally perform this call.

Throws:

java.io.IOException - if an input/output error occurs

encodeChildren()

```
protected void encodeChildren()
    throws IOException
```

Delegate to the `encodeChildren()` method of our corresponding `javax.faces.component.UIComponent0`. This method is called from `doStartTag()`. Normally, delegation occurs unconditionally; however, this method is abstracted out so that advanced tags can conditionally perform this call.

Throws:

```
java.io.IOException - if an input/output error occurs
```

encodeEnd()

```
protected void encodeEnd()
    throws IOException
```

Delegate to the `encodeEnd()` method of our corresponding `javax.faces.component.UIComponent0`. This method is called from `doStartTag()`. Normally, delegation occurs unconditionally; however, this method is abstracted out so that advanced tags can conditionally perform this call.

Throws:

```
java.io.IOException - if an input/output error occurs
```

findComponent()

```
protected javax.faces.component.UIComponent0 findComponent()
    throws JspException
```

Find and return the component, from the component tree, that corresponds to the relative identifier defined by the `id` attribute of this tag. If the value of the `id` attribute is non null, see if there is a component with this `id` in the namespace of the nearest ancestor to the top of the stack that is a naming container. If so, return that component. If the `id` is null, generate a `tagKey` and use it as a key into `tagHash` to locate the component. If component is found, return it. Otherwise create the component and store it in `tagHash` against the generated `tagKey`. `tagHash` is stored as an attribute in root component so that it is available on postback.

Throws:

```
javax.servlet.jsp.JspException - if the specified component cannot be located
javax.servlet.jsp.JspException - if the id is non-null, and no NamingContainer
can be found.
```

findComponentStack()

```
protected java.util.Stack findComponentStack()
```

Locate and return the component stack for this response, creating one if this has not been done already.

getChildIndex()

```
protected int getChildIndex()
```

Returns current value of childIndex.

getComponent()

```
public javax.faces.component.UIComponent getComponent()
```

Return the javax.faces.component.UIComponent₀ instance that is associated with this tag instance. This method is designed to be used by tags nested within this tag, and only returns useful results between the execution of doStartTag() and doEndTag() on this tag instance.

getCreated()

```
public boolean getCreated()
```

Return true if we dynamically created a new component instance during our doStartTag() method.

getDoAfterBodyValue()

```
protected int getDoAfterBodyValue()
    throws JspException
```

Return the flag value that should be returned from the doAfterBody() method when it is called. Subclasses may override this method to return the appropriate value.

Throws:

```
javax.servlet.jsp.JspException - to cause doAfterBody( ) to throw an exception
```

getDoEndValue()

```
protected int getDoEndValue()
    throws JspException
```

Return the flag value that should be returned from the doEnd() method when it is called. Subclasses may override this method to return the appropriate value.

Throws:

```
javax.servlet.jsp.JspException - to cause doEnd( ) to throw an exception
```

getDoStartValue()

```
protected int getDoStartValue()
    throws JspException
```

Return the flag value that should be returned from the `doStart()` method when it is called. Subclasses may override this method to return the appropriate value.

Throws:

`javax.servlet.jsp.JspException` - to cause `doStart()` to throw an exception

getModelReference()

```
public java.lang.String getModelReference()
```

Return the override for the model reference expression.

getNearestEnclosingFacesTag()

```
protected javax.servlet.jsp.tagext.Tag getNearestEnclosingFacesTag()
```

Returns the nearest enclosing `FacesTag`.

getNumChildren()

```
protected int getNumChildren()
```

Returns current value of `numChildren`.

getRendered()

```
public boolean getRendered()
```

Return the override for the `rendered` attribute.

getRendererType()

```
public abstract java.lang.String getRendererType()
```

Return the `rendererType` property that selects the `Renderer` to be used for encoding this component, or `null` to ask the component to render itself directly. Subclasses must override this method to return the appropriate value.

incrementNumChildren()

```
protected void incrementNumChildren()
```

Increments current value of `numChildren`.

isSuppressed()

```
protected boolean isSuppressed()
```

Return true if rendering should be suppressed because some parent component on the stack has been configured with getRendersChildren() as true.

overrideProperties(UIComponent)

```
protected void overrideProperties(javax.faces.component.UIComponent component)
```

Override properties of the specified component if the corresponding properties of this tag handler were explicitly set, and the corresponding attribute of the component is not set.

Tag subclasses that want to support additional override properties must ensure that the base class `overrideProperties()` method is still called. A typical implementation that supports extra properties `foo` and `bar` would look something like this:

```
protected void overrideProperties(UIComponent component) {  
    super.overrideProperties(component);  
    if (foo != null) {  
        component.setAttribute("foo", foo);  
    }  
    if (bar != null) {  
        component.setAttribute("bar", bar);  
    }  
}
```

release()

```
public void release()
```

Release any resources allocated during the execution of this tag handler.

Overrides: `release` in class `TagSupport`

setModelReference(String)

```
public void setModelReference(java.lang.String modelReference)
```

Set an override for the model reference expression.

Parameters:

`modelReference` - The new model reference expression

setParent(Tag)

```
public void setParent(javax.servlet.jsp.tagext.Tag t)
```

Override this method to do three things.

- Call `super.setParent(t)`.

- Find the closest enclosing FacesTag and increment its numChildren counter.
- save our childIndex as enclosingFacesTag.numChildren - 1.

Overrides: setParent in class TagSupport

setRendered(boolean)

```
public void setRendered(boolean rendered)
```

Set an override for the rendered attribute.

Parameters:

rendered - The new value for rendered attribute

setupResponseWriter()

```
protected void setupResponseWriter()
```

Set up the javax.faces.context.ResponseWriter₀ for the current response, if this has not been done already.

javax.faces.webapp FacetTag

Declaration

```
public class FacetTag extends javax.servlet.jsp.tagext.TagSupport  
  
java.lang.Object  
|  
+--javax.servlet.jsp.tagext.TagSupport  
|  
+--javax.faces.webapp.FacetTag
```

All Implemented Interfaces: javax.servlet.jsp.tagext.IterationTag,
java.io.Serializable, javax.servlet.jsp.tagext.Tag

Description

FacetTag is the JSP mechanism for denoting a `javax.faces.component.UIComponent`₀ is to be added as a facet to the component associated with its parent.

A **FacetTag** must have one and only one tag child. This child must be a `FacesTag`₀ instance representing a single `javax.faces.component.UIComponent`₀ instance.

Member Summary

Fields

protected int children ₀	The number of child component tags that have been nested inside this tag.
protected name ₀ java.lang.String	The name of this facet.

Constructors

```
FacetTag()
```

Methods

int doEndTag() ₀	
int doStartTag() ₀	
java.lang.String getName() ₀	
void release() ₀	
void setName(java.lang.String newName) ₀	
void verifySingleChild() ₀	Cause a JspException to be thrown if we have more than one FacesTag ₀ child.

Inherited Member Summary

Fields inherited from interface `IterationTag`

`EVAL_BODY AGAIN`

Fields inherited from interface `Tag`

`EVAL_BODY_INCLUDE, EVAL_PAGE, SKIP_BODY, SKIP_PAGE`

Fields inherited from class `TagSupport`

`id, pageContext`

Methods inherited from class `Object`

`clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(),
toString(), wait(), wait(), wait()`

Methods inherited from class `TagSupport`

`doAfterBody(), findAncestorWithClass(Tag, Class), getId(), getParent(),
getValue(String), getValues(), removeValue(String), setId(String),
 setPageContext(PageContext), setParent(Tag), setValue(String, Object)`

Fields

`children`

`protected int children`

The number of child component tags that have been nested inside this tag.

`name`

`protected java.lang.String name`

The name of this facet. This will be used as the facet name for our `FacesTag` child in our `FacesTag` parent's facet list.

Constructors

`FacetTag()`

`public FacetTag()`

Methods

doEndTag()

```
public int doEndTag()
    throws JspException
```

Overrides: doEndTag in class TagSupport

Throws:

```
javax.servlet.jsp.JspException
```

doStartTag()

```
public int doStartTag()
    throws JspException
```

Overrides: doStartTag in class TagSupport

Throws:

```
javax.servlet.jsp.JspException
```

getName()

```
public java.lang.String getName()
```

release()

```
public void release()
```

Overrides: release in class TagSupport

setName(String)

```
public void setName(java.lang.String newName)
```

verifySingleChild()

```
public void verifySingleChild()
    throws JspException
```

Cause a JspException to be thrown if we have more than one FacesTag₀ child.

Throws:

```
javax.servlet.jsp.JspException - if we have more than one FacesTag0 child.
```

javax.faces.webapp JspResponseWriter

Declaration

```
public final class JspResponseWriter extends ResponseWriterBase  
  
java.lang.Object  
|  
+--java.io.Writer  
|  
+--javax.faces.context.ResponseWriter0  
|  
+--javax.faces.webapp.ResponseWriterBase  
|  
+--javax.faces.webapp.JspResponseWriter
```

Description

JspResponseWriter is a concrete implementation of `javax.faces.context.ResponseWriter0` that wraps a `PageContext` passed to its constructor.

Member Summary

Constructors

```
JspResponseWriter( javax.servlet.jsp.PageContext pageContext )0  
Create a JspResponseWriter that wraps the specified PageContext.
```

Methods

void close() ₀	Close the underlying writer.
void endDocument()	Write whatever text should end a response.
void endElement(java.lang.String name)	Write the end of an element, after closing any open element created by a call to startElement().
void flush() ₀	Flush the underlying writer's buffer, if any.
void startDocument()	Write whatever text should begin a response.
void startElement(java.lang.String name)	Write the start of an element, up to and including the element name.
void write(char[] cbuf) ₀	Write an array of characters.

Member Summary

```
void write(char[] cbuf, int off, int len)
        Write a portion of an array of characters.
void write(int c)
        Write a single character from the low-order 16 bits of the specified integer.
void write(java.lang.String s)
        Write a String of characters.
void write(java.lang.String s, int off, int len)
        Write a portion of a String of characters.
void writeAttribute(java.lang.String name, java.lang.Object value)
        Write an attribute name and corresponding value (after converting that text to a String if necessary), after escaping it properly.
void writeComment(java.lang.Object comment)
        Write a comment containing the specified text, after converting that text to a String if necessary.
void writeText(char text)
        Write a single character, after escaping it properly.
void writeText(char[] text)
        Write text from a character array, after escaping it properly.
void writeText(char[] text, int off, int len)
        Write text from a character array, after escaping it properly for this method.
void writeText(java.lang.Object text)
        Write an object (after converting it to a String, if necessary), after escaping it properly.
void writeURIAttribute(java.lang.String name, java.lang.Object value)
        Write a URI attribute name and corresponding value (after converting that text to a String if necessary), after encoding it properly (for example, '%' encoded for HTML).
```

Inherited Member Summary**Fields inherited from class `Writer`**

lock

Methods inherited from class `Object``clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(),
toString(), wait(), wait(), wait()`

Constructors

JspResponseWriter(PageContext)

```
public JspResponseWriter(javax.servlet.jsp.PageContext pageContext)
```

Create a JspResponseWriter that wraps the specified PageContext.

Parameters:

writer - The JspWriter to be wrapped

Throws:

NullPointerException - if pageContext is null

Methods

close()

```
public void close()
                  throws IOException
```

Close the underlying writer.

Overrides: close in class Writer

Throws:

java.io.IOException - if an input/output error occurs

endDocument()

```
public void endDocument()
                        throws IOException
```

Write whatever text should end a response. If there is an open element that has been created by a call to startElement(), that element will be closed first.

Overrides: endDocument₀ in class ResponseWriter₀

Throws:

java.io.IOException - if an input/output error occurs

endElement(String)

```
public void endElement(java.lang.String name)
                        throws IOException
```

Write the end of an element, after closing any open element created by a call to startElement().

Overrides: endElement₀ in class ResponseWriter₀

Parameters:

name - Name of the element to be ended

Throws:

java.io.IOException - if an input/output error occurs

NullPointerException - if name is null

flush()

```
public void flush()
                  throws IOException
```

Flush the underlying writer's buffer, if any.

Overrides: flush in class Writer

Throws:

java.io.IOException - if an input/output error occurs

startDocument()

```
public void startDocument()
                  throws IOException
```

Write whatever text should begin a response.

Overrides: startDocument₀ in class ResponseWriter₀

Throws:

java.io.IOException - if an input/output error occurs

startElement(String)

```
public void startElement(java.lang.String name)
                  throws IOException
```

Write the start of an element, up to and including the element name. Once this method has been called, clients can call writeAttribute() or writeURIAttribute() method to add attributes and corresponding values. The starting element will be closed (that is, the trailing '>' character added) on any subsequent call to startElement(), writeComment(), writeText(), endElement(), or endDocument().

Overrides: startElement₀ in class ResponseWriter₀

Parameters:

name - Name of the element to be started

Throws:

java.io.IOException - if an input/output error occurs
NullPointerException - if name is null

write(char[])

```
public void write(char[] cbuf)
    throws IOException
```

Write an array of characters.

Overrides: write in class Writer

Parameters:

cbuf - An array of characters to be written

Throws:

java.io.IOException - if an input/output error occurs

write(char[], int, int)

```
public void write(char[] cbuf, int off, int len)
    throws IOException
```

Write a portion of an array of characters.

Overrides: write in class Writer

Parameters:

cbuf - An array of characters to be written

off - Starting offset (zero-relative)

len - Number of characters to write

Throws:

java.io.IOException - if an input/output error occurs

write(int)

```
public void write(int c)
    throws IOException
```

Write a single character from the low-order 16 bits of the specified integer.

Overrides: write in class Writer

Parameters:

c - An int specifying the character to be written

Throws:

java.io.IOException - if an input/output error occurs

write(String)

```
public void write(java.lang.String s)
                  throws IOException
```

Write a String of characters.

Overrides: write in class Writer

Parameters:

s - A String to be written

Throws:

java.io.IOException - if an input/output error occurs

write(String, int, int)

```
public void write(java.lang.String s, int off, int len)
                  throws IOException
```

Write a portion of a String of characters.

Overrides: write in class Writer

Parameters:

s - A String of characters to be written

off - Starting offset (zero-relative)

len - Number of characters to write

Throws:

java.io.IOException - if an input/output error occurs

writeAttribute(String, Object)

```
public void writeAttribute(java.lang.String name, java.lang.Object value)
                           throws IOException
```

Write an attribute name and corresponding value (after converting that text to a String if necessary), after escaping it properly. This method may only be called after a call to startElement(), and before the opened element has been closed.

Overrides: writeAttribute₀ in class ResponseWriter₀

Parameters:

name - Attribute name to be added

value - Attribute value to be added

Throws:

java.lang.IllegalStateException - if this method is called when there is no currently open element

```
java.io.IOException - if an input/output error occurs  
NullPointerException - if name or value is null
```

writeComment(Object)

```
public void writeComment(java.lang.Object comment)  
throws IOException
```

Write a comment containing the specified text, after converting that text to a String if necessary. If there is an open element that has been created by a call to `startElement()`, that element will be closed first.

Overrides: `writeComment` in class `ResponseWriter`

Parameters:

comment - Text content of the comment

Throws:

```
java.io.IOException - if an input/output error occurs  
NullPointerException - if comment is null
```

writeText(char)

```
public void writeText(char text)  
throws IOException
```

Write a single character, after escaping it properly. If there is an open element that has been created by a call to `startElement()`, that element will be closed first.

Overrides: `writeText` in class `ResponseWriter`

Parameters:

text - Text to be written

Throws:

```
java.io.IOException - if an input/output error occurs
```

writeText(char[])

```
public void writeText(char[] text)  
throws IOException
```

Write text from a character array, after escaping it properly. This is equivalent to calling `writeText(c, 0, c.length)`. If there is an open element that has been created by a call to `startElement()`, that element will be closed first.

Overrides: `writeText` in class `ResponseWriter`

Parameters:

text - Text to be written

Throws:

java.io.IOException - if an input/output error occurs

NullPointerException - if text is null

writeText(char[], int, int)

```
public void writeText(char[] text, int off, int len)
    throws IOException
```

Write text from a character array, after escaping it properly for this method. If there is an open element that has been created by a call to `startElement()`, that element will be closed first.

Overrides: `writeText` in class `ResponseWriter`

Parameters:

text - Text to be written

off - Starting offset (zero-relative)

len - Number of characters to be written

Throws:

java.lang.IndexOutOfBoundsException - if the calculated starting or ending position is outside the bounds of the character array

java.io.IOException - if an input/output error occurs

NullPointerException - if text is null

writeText(Object)

```
public void writeText(java.lang.Object text)
    throws IOException
```

Write an object (after converting it to a String, if necessary), after escaping it properly. If there is an open element that has been created by a call to `startElement()`, that element will be closed first.

Overrides: `writeText` in class `ResponseWriter`

Parameters:

text - Text to be written

Throws:

java.io.IOException - if an input/output error occurs

NullPointerException - if text is null

writeURIAttribute(String, Object)

```
public void writeURIAttribute(java.lang.String name, java.lang.Object value)
    throws IOException
```

Write a URI attribute name and corresponding value (after converting that text to a String if necessary), after encoding it properly (for example, '%' encoded for HTML). This method may only be called after a call to `startElement()`, and before the opened element has been closed.

Overrides: `writeURIAttribute` in class `ResponseWriter`

Parameters:

name - Attribute name to be added

value - Attribute value to be added

Throws:

`java.lang.IllegalStateException` - if this method is called when there is no currently open element

`java.io.IOException` - if an input/output error occurs

`NullPointerException` - if name or value is null

javax.faces.webapp ServletResponseWriter

Declaration

```
public final class ServletResponseWriter extends ResponseWriterBase

java.lang.Object
 |
 +--java.io.Writer
   |
   +--javax.faces.context.ResponseWriter0
     |
     +--javax.faces.webapp.ResponseWriterBase
       |
       +--javax.faces.webapp.HttpServletResponseWriter
```

Description

ServletResponseWriter is a concrete implementation of `javax.faces.context.ResponseWriter0` that wraps a `PrintWriter` passed to its constructor.

Member Summary

Constructors

```
ServletResponseWriter(java.io.PrintWriter writer)0
    Create a HttpServletResponseWriter that wraps the specified PrintWriter.
```

Methods

```
void close()0
    Close the underlying writer.
void endDocument()
    Write whatever text should end a response.
void endElement(java.lang.String name)
    Write the end of an element, after closing any open element created by a call to
    startElement().
void flush()0
    Flush the underlying writer's buffer, if any.
void startDocument()
    Write whatever text should begin a response.
void startElement(java.lang.String name)
    Write the start of an element, up to and including the element name.
void write(char[] cbuf)0
    Write an array of characters.
```

Member Summary

```
void write(char[] cbuf, int off, int len)
          Write a portion of an array of characters.
void write(int c)
          Write a single character from the low-order 16 bits of the specified integer.
void write(java.lang.String s)
          Write a String of characters.
void write(java.lang.String s, int off, int len)
          Write a portion of a String of characters.
void writeAttribute(java.lang.String name, java.lang.Object value)
          Write an attribute name and corresponding value (after converting that text to a String if necessary), after escaping it properly.
void writeComment(java.lang.Object comment)
          Write a comment containing the specified text, after converting that text to a String if necessary.
void writeText(char text)
          Write a single character, after escaping it properly.
void writeText(char[] text)
          Write text from a character array, after escaping it properly.
void writeText(char[] text, int off, int len)
          Write text from a character array, after escaping it properly for this method.
void writeText(java.lang.Object text)
          Write an object (after converting it to a String, if necessary), after escaping it properly.
void writeURIAttribute(java.lang.String name, java.lang.Object value)
          Write a URI attribute name and corresponding value (after converting that text to a String if necessary), after encoding it properly (for example, '%' encoded for HTML).
```

Inherited Member Summary**Fields inherited from class `Writer`**

lock

Methods inherited from class `Object`

```
clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(),
toString(), wait(), wait(), wait()
```

Constructors

ServletResponseWriter(PrintWriter)

```
public ServletResponseWriter(java.io.PrintWriter writer)
```

Create a ServletResponseWriter that wraps the specified PrintWriter.

Parameters:

writer - The PrintWriter to be wrapped

Throws:

NullPointerException - if writer is null

Methods

close()

```
public void close()
                  throws IOException
```

Close the underlying writer.

Overrides: close in class Writer

Throws:

java.io.IOException - if an input/output error occurs

endDocument()

```
public void endDocument()
                  throws IOException
```

Write whatever text should end a response. If there is an open element that has been created by a call to startElement(), that element will be closed first.

Overrides: endDocument₀ in class ResponseWriter₀

Throws:

java.io.IOException - if an input/output error occurs

endElement(String)

```
public void endElement(java.lang.String name)
                  throws IOException
```

Write the end of an element, after closing any open element created by a call to startElement().

Overrides: endElement₀ in class ResponseWriter₀

Parameters:

name - Name of the element to be ended

Throws:

java.io.IOException - if an input/output error occurs

NullPointerException - if name is null

flush()

```
public void flush()
                  throws IOException
```

Flush the underlying writer's buffer, if any.

Overrides: flush in class Writer

Throws:

java.io.IOException - if an input/output error occurs

startDocument()

```
public void startDocument()
                  throws IOException
```

Write whatever text should begin a response.

Overrides: startDocument₀ in class ResponseWriter₀

Throws:

java.io.IOException - if an input/output error occurs

startElement(String)

```
public void startElement(java.lang.String name)
                  throws IOException
```

Write the start of an element, up to and including the element name. Once this method has been called, clients can call writeAttribute() or writeURIAttribute() method to add attributes and corresponding values. The starting element will be closed (that is, the trailing '>' character added) on any subsequent call to startElement(), writeComment(), writeText(), endElement(), or endDocument().

Overrides: startElement₀ in class ResponseWriter₀

Parameters:

name - Name of the element to be started

Throws:

```
java.io.IOException - if an input/output error occurs  
NullPointerException - if name is null
```

write(char[])

```
public void write(char[] cbuf)  
    throws IOException
```

Write an array of characters.

Overrides: write in class Writer

Parameters:

cbuf - An array of characters to be written

Throws:

```
java.io.IOException - if an input/output error occurs
```

write(char[], int, int)

```
public void write(char[] cbuf, int off, int len)  
    throws IOException
```

Write a portion of an array of characters.

Overrides: write in class Writer

Parameters:

cbuf - An array of characters to be written

off - Starting offset (zero-relative)

len - Number of characters to write

Throws:

```
java.io.IOException - if an input/output error occurs
```

write(int)

```
public void write(int c)  
    throws IOException
```

Write a single character from the low-order 16 bits of the specified integer.

Overrides: write in class Writer

Parameters:

c - An int specifying the character to be written

Throws:

```
java.io.IOException - if an input/output error occurs
```

write(String)

```
public void write(java.lang.String s)
                 throws IOException
```

Write a String of characters.

Overrides: write in class Writer

Parameters:

s - A String to be written

Throws:

java.io.IOException - if an input/output error occurs

write(String, int, int)

```
public void write(java.lang.String s, int off, int len)
                 throws IOException
```

Write a portion of a String of characters.

Overrides: write in class Writer

Parameters:

s - A String of characters to be written

off - Starting offset (zero-relative)

len - Number of characters to write

Throws:

java.io.IOException - if an input/output error occurs

writeAttribute(String, Object)

```
public void writeAttribute(java.lang.String name, java.lang.Object value)
                           throws IOException
```

Write an attribute name and corresponding value (after converting that text to a String if necessary), after escaping it properly. This method may only be called after a call to startElement(), and before the opened element has been closed.

Overrides: writeAttribute₀ in class ResponseWriter₀

Parameters:

name - Attribute name to be added

value - Attribute value to be added

Throws:

java.lang.IllegalStateException - if this method is called when there is no currently open element

```
java.io.IOException - if an input/output error occurs  
NullPointerException - if name or value is null
```

writeComment(Object)

```
public void writeComment(java.lang.Object comment)  
    throws IOException
```

Write a comment containing the specified text, after converting that text to a String if necessary. If there is an open element that has been created by a call to `startElement()`, that element will be closed first.

Overrides: `writeComment` in class `ResponseWriter`

Parameters:

comment - Text content of the comment

Throws:

```
java.io.IOException - if an input/output error occurs  
NullPointerException - if comment is null
```

writeText(char)

```
public void writeText(char text)  
    throws IOException
```

Write a single character, after escaping it properly. If there is an open element that has been created by a call to `startElement()`, that element will be closed first.

Overrides: `writeText` in class `ResponseWriter`

Parameters:

text - Text to be written

Throws:

```
java.io.IOException - if an input/output error occurs
```

writeText(char[])

```
public void writeText(char[] text)  
    throws IOException
```

Write text from a character array, after escaping it properly. This is equivalent to calling `writeText(c, 0, c.length)`. If there is an open element that has been created by a call to `startElement()`, that element will be closed first.

Overrides: `writeText` in class `ResponseWriter`

Parameters:

text - Text to be written

Throws:

java.io.IOException - if an input/output error occurs

NullPointerException - if text is null

writeText(char[], int, int)

```
public void writeText(char[] text, int off, int len)
                      throws IOException
```

Write text from a character array, after escaping it properly for this method. If there is an open element that has been created by a call to `startElement()`, that element will be closed first.

Overrides: `writeText()` in class `ResponseWriter`

Parameters:

text - Text to be written

off - Starting offset (zero-relative)

len - Number of characters to be written

Throws:

java.lang.IndexOutOfBoundsException - if the calculated starting or ending position is outside the bounds of the character array

java.io.IOException - if an input/output error occurs

NullPointerException - if text is null

writeText(Object)

```
public void writeText(java.lang.Object text)
                      throws IOException
```

Write an object (after converting it to a String, if necessary), after escaping it properly. If there is an open element that has been created by a call to `startElement()`, that element will be closed first.

Overrides: `writeText()` in class `ResponseWriter`

Parameters:

text - Text to be written

Throws:

java.io.IOException - if an input/output error occurs

NullPointerException - if text is null

writeURIAttribute(String, Object)

```
public void writeURIAttribute(java.lang.String name, java.lang.Object value)
    throws IOException
```

Write a URI attribute name and corresponding value (after converting that text to a String if necessary), after encoding it properly (for example, '%' encoded for HTML). This method may only be called after a call to `startElement()`, and before the opened element has been closed.

Overrides: `writeURIAttribute` in class `ResponseWriter`

Parameters:

`name` - Attribute name to be added

`value` - Attribute value to be added

Throws:

`java.lang.IllegalStateException` - if this method is called when there is no currently open element

`java.io.IOException` - if an input/output error occurs

`NullPointerException` - if `name` or `value` is `null`

javax.faces.webapp ValidatorTag

Declaration

```
public class ValidatorTag extends javax.servlet.jsp.tagext.TagSupport  
  
java.lang.Object  
|  
+--javax.servlet.jsp.tagext.TagSupport  
|  
+--javax.faces.webapp.ValidatorTag
```

All Implemented Interfaces: javax.servlet.jsp.tagext.IterationTag,
java.io.Serializable, javax.servlet.jsp.tagext.Tag

Description

ValidatorTag is a base class for all JSP custom actions that create and register a **Validator** instance on the `javax.faces.component.UIComponent0`, associated with our most immediate surrounding instance of a tag whose implementation class is a subclass of `FacesTag0`. To avoid creating duplicate instances when a page is redisplayed, creation and registration of a `javax.faces.validator.Validator0` occurs **only** if the corresponding `javax.faces.component.UIComponent0` was created (by the owning `FacesTag0`) during the execution of the current page.

This class may be used directly to implement a generic validator registration tag (based on the fully qualified Java class name specified by the `type` attribute), or as a base class for tag instances that support specific `javax.faces.validator.Validator0` subclasses.

Subclasses of this class must implement the `createValidator()` method, which creates and returns a `javax.faces.validator.Validator0` instance. Any configuration properties that specify the limits to be enforced by this `javax.faces.validator.Validator0` must have been set by the `createValidator()` method. Generally, this occurs by copying corresponding attribute values on the tag instance.

This tag creates no output to the page currently being created. It is used solely for the side effect of `javax.faces.validator.Validator0` creation.

Member Summary

Constructors

`ValidatorTag()`

Member Summary

Methods

```
protected createValidator()
javax.faces.validator.      Create and return a new javax.faces.validator.Validator0 to be
    Validator      registered on our surrounding javax.faces.component.UIComponent0.
int doStartTag()
    Create a new instance of the specified javax.faces.validator.Validator0
    class, and register it with the javax.faces.component.UIComponent0
    instance associated with our most immediately surrounding FacesTag0 instance, if
    the javax.faces.component.UIComponent0 instance was created by this
    execution of the containing JSP page.
void release()
    Release references to any acquired resources.
void setType(java.lang.String type)
    Set the fully qualified class name of the
    javax.faces.validator.Validator0 instance to be created.
```

Inherited Member Summary

Fields inherited from interface IterationTag

EVAL_BODY AGAIN

Fields inherited from interface Tag

EVAL_BODY_INCLUDE, EVAL_PAGE, SKIP_BODY, SKIP_PAGE

Fields inherited from class TagSupport

id, pageContext

Methods inherited from class Object

clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(),
toString(), wait(), wait(), wait()

Methods inherited from class TagSupport

doAfterBody(), doEndTag(), findAncestorWithClass(Tag, Class), getId(), getParent(),
getValue(String), getValues(), removeValue(String), setId(String),
setPageContext(PageContext), setParent(Tag), setValue(String, Object)

Constructors

ValidatorTag()

```
public ValidatorTag()
```

Methods

createValidator()

```
protected javax.faces.validator.Validator_ createValidator()
    throws JspException
```

Create and return a new `javax.faces.validator.Validator0` to be registered on our surrounding `javax.faces.component.UIComponent0`.

Throws:

`javax.servlet.jsp.JspException` - if a new instance cannot be created

doStartTag()

```
public int doStartTag()
    throws JspException
```

Create a new instance of the specified `javax.faces.validator.Validator0` class, and register it with the `javax.faces.component.UIComponent0` instance associated with our most immediately surrounding `FacesTag0` instance, if the `javax.faces.component.UIComponent0` instance was created by this execution of the containing JSP page.

Overrides: `doStartTag` in class `TagSupport`

Throws:

`javax.servlet.jsp.JspException` - if a JSP error occurs

release()

```
public void release()
```

Release references to any acquired resources.

Overrides: `release` in class `TagSupport`

setType(String)

```
public void setType(java.lang.String type)
```

Set the fully qualified class name of the `javax.faces.validator.Validator` instance to be created.

Parameters:

`type` - The new class name