Web Application Framework Component Reference Guide

Sun Java™ Studio Enterprise 7 2004Q4

Sun Microsystems, Inc.
www.sun.com

Part No. 819-0725-10
December 2004, Revision A

Submit comments about this document at: http://www.sun.com/hwdocs/feedback

Copyright © 2004 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A. All rights reserved.

U.S. Government Rights - Commercial software. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements.

This distribution may include materials developed by third parties.Sun, Sun Microsystems, the Sun logo and Java are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon architecture developed by Sun Microsystems, Inc.

UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Products covered by and information contained in this service manual are controlled by U.S. Export Control laws and may be subject to the export or import laws in other countries. Nuclear, missile, chemical biological weapons or nuclear maritime end uses or end users, whether direct or indirect, are strictly prohibited. Export or reexport to countries subject to U.S. embargo or to entities identified on U.S. export exclusion lists, including, but not limited to, the denied persons and specially designated nationals lists is strictly prohibited.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright © 2004 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, Etats-Unis. Tous droits réservés.

Cette distribution peut comprendre des composants développés par des tierces parties.Sun, Sun Microsystems, le logo Sun et Java sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays.

Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

UNIX est une marque déposée aux Etats-Unis et dans d'autres pays et licenciée exlusivement par X/Open Company, Ltd.

Les produits qui font l'objet de ce manuel d'entretien et les informations qu'il contient sont regis par la legislation americaine en matiere de controle des exportations et peuvent etre soumis au droit d'autres pays dans le domaine des exportations et importations. Les utilisations finales, ou utilisateurs finaux, pour des armes nucleaires, des missiles, des armes biologiques et chimiques ou du nucleaire maritime, directement ou indirectement, sont strictement interdites. Les exportations ou reexportations vers des pays sous embargo des Etats-Unis, ou vers des entites figurant sur les listes d'exclusion d'exportation americaines, y compris, mais de maniere non exclusive, la liste de personnes qui font objet d'un ordre de ne pas participer, d'une facon directe ou indirecte, aux exportations des produits ou des services qui sont regi par la legislation americaine en matiere de controle des exportations et la liste de ressortissants specifiquement designes, sont rigoureusement interdites.LA

DOCUMENTATION EST FOURNIE "EN L'ETAT" ET TOUTES AUTRES CONDITIONS, DECLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUES, DANS LA MESURE AUTORISEE PAR LA LOI APPLICABLE, Y COMPRIS NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE A LA QUALITE MARCHANDE, A L'APTITUDE A UNE UTILISATION PARTICULIERE OU A L'ABSENCE DE CONTREFACON.

Adobe PostScript

Contents

Before You Begin

The *Web Application Framework Component Reference Guide* introduces the components in the Web Application Framework Library. The components fall into four basic groups: Visual components, Model Components, Command Components, and Non-Visual Components.

Before You Read This Book

Before starting, you should be familiar with concepts used in building web applications using existing J2EE web technologies, such as servlets and JavaServlet Pages™ (JSP™ pages). You should also be familiar with the Web Application Framework architecture and the Sun Java Studio Enterprise 7 development environment (hereafter referred to as the IDE) by referring to the related Web Application Framework documentation mentioned later in this chapter.

The following resources can provide additional information:

- *Java 2 Platform, Enterprise Edition Specification*
  `http://java.sun.com/j2ee/download.html#platformspec`
- *The J2EE Tutorial*
  `http://java.sun.com/j2ee/tutorial`
- *Java Servlet Specification Version 2.3*
  `http://java.sun.com/products/servlet/download.html#specs`
- *JavaServer Pages Specification Version 1.2*
  `http://java.sun.com/products/jsp/download.html#specs`

Note – Sun is not responsible for the availability of third-party Web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused by or in connection with the use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

Typographic Conventions

Typeface	Meaning	Examples
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output	Edit your `.cvspass` file. Use `DIR` to list all files. `Search is complete.`
`AaBbCc123`	What you type, when contrasted with on-screen computer output	`> login` `Password:`
AaBbCc123	Book titles, new words or terms, words to be emphasized	Read Chapter 6 in the *User's Guide*. These are called *class* options. You *must* save your changes.
AaBbCc123	Command-line variable; replace with a real name or value	To delete a file, type **`DEL`** *filename*.

Related Documentation

Java Studio Enterprise documentation includes books and tutorials delivered in Acrobat Reader (PDF) format, release notes, online help, and tutorials delivered in HTML format.

Documentation Available Online

The documents described in this section are available from the `docs.sun.com`SM web site and from the Documentation link from the Sun Java Studio Enterprise Developers Source portal (`http://developers.sun.com/jsenterprise`).

The `docs.sun.com` web site (`http://docs.sun.com`) enables you to read, print, and buy Sun Microsystems manuals through the Internet.

- *Sun Java Studio Enterprise 7 Release Notes* - part no. 819-0905-10

  Describes last-minute release changes and technical notes.

- *Sun Java Studio Enterprise 7 Installation Guide* (PDF format) - part no. 817-7971-10

  Describes how to install the Sun Java Studio Enterprise 7 integrated development environment (IDE) on each supported platform and includes other pertinent information, such as system requirements, upgrade instructions, server information, command-line switches, installed subdirectories, database integration, and information on how to use the Update Center.

- *Building J2EE Applications* - part no. 819-0819-10

  Describes how to assemble EJB modules and web modules into a J2EE application and how to deploy and run a J2EE application.

- Web Application Framework documentation (PDF format)
  - *Web Application Framework Component Author's Guide* - part no. 819-0724-10

    Describes the Web Application Framework component architecture and the process to design, create, and distribute new components.

  - *Web Application Framework Component Reference Guide* - part no. 819-0725-10

    Describes the components available in the Web Application Framework Library.

  - *Web Application Framework Overview* - part no. 819-0726-10

    Introduces the Web Application Framework and what it is, how it works, and what sets it apart from other application frameworks.

  - *Web Application Framework Tutorial*- part no. 819-0727-10

    Introduces the mechanics and techniques to build a web application using the Web Application Framework tools.

  - *Web Application Framework Developer's Guide* - part no. 819-0728-10

    Provides the steps to create and use application components that can be assembled to develop an application using the Web Application Framework and explains how to deploy the application in most J2EE containers.

- *Web Application Framework IDE Guide* - part no. 819-0729-10

  Describes the various parts of the Sun Java Studio Enterprise 7 2004Q4 IDE and emphasizes the use of the visual tools for developing a Web Application Framework application.

- *Web Application Framework Tag Library Reference* - part no. 819-0730-10

  Gives a brief introduction to the Web Application Framework tag library, as well as a comprehensive reference to the tags available within the library.

Tutorials

Sun Java Studio Enterprise 7 tutorials help you understand the features of the IDE. Each tutorial provides techniques and code samples that you can use or modify in developing more substantial applications. All tutorials illustrate deployment with Sun Java System Application Server.

All tutorials are available from the Tutorials and Code Camps link off the Developers Source portal, which you can access from within the IDE by choosing Help > Examples and Tutorials.

- **QuickStart guides** provide an introduction to the Sun Java Studio IDE. Start with a QuickStart tutorial if you are either new to the Sun Java Studio IDE or want a quick introduction to a particular feature. These tutorials describe how to develop simple web and J2EE applications, generate web services, and how to get started with UML modeling and Refactoring. QuickStarts take minutes to complete.
- **Tutorials** focus on a single feature of the Sun Java Studio IDE. Try these if you are interested in the details of a particular feature. Some tutorials build an application from the ground up, while others build on provided source files, depending on the focus of the example. You can complete a tutorial in an hour or less.
- **Narrated Tutorials** use video to illustrate a feature or technique. Try a narrated tutorials for a visual overview of the IDE or an in-depth presentation of a particular feature. You can complete a narrated tutorial in a few minutes. You can also start and stop a narrated tutorial at any point you wish.

Online Help

Online help is available in the Sun Java Studio Enterprise 7 IDE. You can open help by pressing the help key (F1 in Microsoft Windows environments, Help key in the Solaris environment), or by choosing Help → Contents. Either action displays a list of help topics and a search facility.

Documentation in Accessible Formats

The documentation is provided in accessible formats that are readable by assistive technologies for users with disabilities. You can find accessible versions of documentation as described in the following table.

Type of Documentation	Format and Location of Accessible Version
Books and tutorials	HTML at `http://docs.sun.com`
Tutorials	HTML at the Examples and Code Camps link from the Developers Source portal at `http://developers.sun.com/jsenterprise`
Release notes	HTML at `http://docs.sun.com`

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. Email your comments to Sun at this address:

`docfeedback@sun.com`

Please include the book's title (*Web Application Framework Component Reference Guide*'s) and its part number (819-0725-10) in the subject line of your email.

CHAPTER 1

Component Overview

The components in the Web Application Framework Component Library fall into four basic groups: Visual Components, Model Components, Command Components, and Non-Visual Components. See the following sections for more information.

The IDE supports the use of both *extensible* and *non-extensible* visual components.

Extensible components are components that can be subclassed. Subclassing of extensible components is transparently facilitated by the IDE. The Web Application Framework IDE wizards automatically create an application-specific class which extends the component base class.

Non-extensible components, visual and non-visual alike, are components that are not normally subclassed in the course of Web Application Framework IDE usage. When a new non-extensible visual component is selected from the component palette, a named instance is created rather than a new subtype.

Visual Components

Visual components are components that developers use to assemble a user interface for an application.

Extensible Visual Components

Component Name	Description
Basic Container View (Pagelet)	A View that can contain other Views.
Basic Tiled View	Aspecialization type of ContainerView that can present its child View components in a number of repeated tiles, or repeated regions.
Basic Tree View	A specialization of ContainerView that helps present information in a tree format.
Basic ViewBean (Page)	A specialization of ContainerView that can serve as the top, or root, of a View component hierarchy.

Non-Extensible Visual Components

Component Name	Description
Button	A command field that is rendered as a button.
Check Box	A component that provides a mutually exclusive, two-state field value: true/false, yes/no, on/off, A/B, etc.
Combo Box	A choice component that provides a list of choices presented in a drop-down list style interface.
Data-Driven Combo Box	A choice component that can have its choices populated from a model component. It provides a list of choices presented in a drop-down list style interface.
Data-Driven List Box	A choice component that can have its choices populated from a model component. It presents its list of choices in a list box. Multiple choices, if allowed, can be selected by the user.
Data-Driven Radio Buttons	A choice component that can have its choices populated from a model component. It presents its list of choices as a group of radio buttons.
Dataset Locator	Displays the record of the displayed data (e.g. Records 1 to 10 of 53).
Dataset Navigator	A set of four command fields that enable pagination (navigation) control over a set of data through a container view, like a BasicViewBean, a BasicTiledView or a BasicContainerView.
Date View	A visual display field component which displays the date whose fields (month, day, year) are represented by Combo Boxes. Also includes a javascript mini-calendar popup.
DateTime View	A visual display field component which displays the date whose fields (month, day, year, hour, minute, etc.) are represented by Combo Boxes. Also includes a javascript mini-calendar popup.
File Upload	A component that provides a way for users to send files to the server.

Component Name	Description
Go To Page Link	A HREF display field component which is preconfigured to use a Goto View Bean command descriptor.
Hidden Field	A component that provides a way to embed non-visible data in a page so that it is sent back to the server when the surrounding form is posted.
Hyperlink (HREF)	A command field that is rendered as a hyperlink.
Image	A component that allows an image to be displayed on a page.
List Box	A choice component that presents its list of choices in a list box. Multiple choices, if allowed, can be selected by the user.
Masked Text Field	A single-line, free-form text input field.
Menu	A View component that allows user to design drop-down menu functionality into their web applications.
Password Field	A component that provides a way for the user to enter text without showing the characters he or she has entered.
Radio Buttons	A choice component that presents its list of choices as a group of radio buttons.
Static Breadcrumb	Displays the current page's context within the site structure.
Static Text Field	A component that displays read-only text or markup.s
Text Field	A single-line, free-form text input field.
Text Area	A multi-line, free-form text input field.
Time View	A visual display field component which displays the time whose fields (hour, minute, etc.) are represented by Combo Boxes.
Validating Text Field	A single-line, free-form text input field with the ability to validate the user-supplied text using an associated validation component.
Validating Text Area	A multi-line, free-form text input field with the ability to validate the user-supplied text using an associated validation component.

Model Components

Model components are components that act as a business delegate or a data proxy to an arbitrary data store (Java class, CORBA object, EJB, database, mainframe, ERP system, transaction processor, etc.).

Component Name	Description
Bean Adapter Model	A model that uses one or more JavaBeans as its backing datastore.
Client Session Model	Uses the JATO client session as its backing data store.
Custom Model	An arbitrary implementation of the `com.iplanet.model.Model` interface.

Component Name	Description
Custom Tree Model	A model that provide access to data stores that employ a hierarchical (tree or directory like) data structure, like XML documents, LDAP repositories, or file systems.
Directory Search Model	Allows us to uses a Model as a backing store for the LDAP Query's Result Set.
HTTP Session Model	A model that uses the HTTP session as its backing datastore.
JDBC ResultSet Adapter Model	Will adapt to the ResultSet thats being passed to it. It allows the user to set the Field Names and Types during the design time.
JDBC SQL Query Model	A model that uses one or more RDBMS tables as the backing datastore.
JDBC Stored Procedure Model	A model that executes a stored procedure.
Object Adapter Model	A model that provides access to any object's fields, bean properties, and/or methods using path expressions that specify *deep* access to object members.
Resource Bundle Model	A model that uses a resource bundle as its backing datastore.
Simple Custom Model	A model that provides a foundation for an arbitrary new model which requires advanced dataset management and pagination support.
Web Service Model	A model that allows developers to easily execute Web service operations.

Command Components

Command components encapsulate arbitrary behavior. Typically, command components encapsulate request handling logic or controller functionality. Command fields (buttons, HREFs) are the primary consumers of command components.

Component Name	Description
Basic Command	An arbitrary controller or request handler component.
Command Chain	A command that links together two or more command components to be invoked in sequence

Non-Visual Components

Component Name	Description
Application Attribute Factory	A factory that acquires an object from application scope.
Execute Model and Goto Page Command	Configures an instance of `com.iplanet.jato.view.command.ExecuteAndForwardCommand`.
Execute Model Command	Configures an instance of `com.iplanet.jato.view.command.ExecuteModelCommand`.
Forward Command	Configures an instance of `com.iplanet.jato.view.command.ForwardCommand`.
Goto ViewBean Command	Configures an instance of `com.iplanet.jato.view.command.GotoViewBeanCommand`.
Include Command	Configures an instance of `com.iplanet.jato.view.command.RedirectCommand`.
Redirect Command	Configures an instance of `com.iplanet.jato.view.command.IncludeCommand`.
Regular Expression Validator	A validator that validates based on successful conversion to a specified type.
Request Attribute Factory	A factory that acquires an object from request scope.
Session Attribute Factory	A factory that acquires an object from session scope.
Simple Choice	A simple Choice implementation.
Model Reference	Configures an instance of `com.iplanet.jato.model.SimpleModelReference`.
Type Validator	A simple validator that uses JDK 1.4 regular expressions to validate a value.
User-Defined Command	Configures an instance of an arbitrary implementation of `com.iplanet.jato.command.Command`.
WebAction Command	Configures an instance of `com.iplanet.jato.view.command.ExecuteAndForwardCommand`.

Component Reference

Note Legend:

Req = Required property

Dependent = Dependent property (for example, a property that is dependent on the value of another property)

CHAPTER 2

Basic Container View (Pagelet)

The Basic Container View component is also referred to as a pagelet component. It is analogous to panel components in other visual development environments, providing a way to group a set of contained components so that they can be manipulated as a unit. Container views also form the basis for most complex components (both distributable and non-distributable), which can be reused by (contained in, parented by) other page and pagelet components.

Property Name	Description	Notes
Auto Deleting Models	A list of *deleting type* models that will be have their `delete(...)` method invoked when the *delete WebAction* is invoked for this container view. The listed models must implement the `com.iplanet.jato.model.DeletingModel` interface.	
Auto Executing Models	A list of *executing type* models that will be have their `execute(...)` method invoked when the execute WebAction is invoked for this container view. The listed models must implement the `com.iplanet.jato.model.ExecutingModel` interface.	
Auto Inserting Models	A list of *inserting type* models that will be have their `insert(...)` method invoked when the insert WebAction is invoked for this container view. The listed models must implement the `com.iplanet.jato.model.InsertingModel` interface.	
Auto Retrieving Models	A list of *retrieving type* models that will be have their `retrieve(...)` method invoked when the retrieve WebAction is invoked for this container view. The listed models must implement the `com.iplanet.jato.model.RetrievingModel` interface.	
Auto Updating Models	A list of *updating type* models that will be have their `update(...)` method invoked when the *update WebAction* is invoked for this container view. The listed models must implement the `com.iplanet.jato.model.UpdatingModel` interface.	

Property Name	Description	Notes
Name	The class name of the component.	Req
Validation Exception Handler	Specifies how a validation exception should be handled: by the `handleValidationException(CommandEvent)` event, or with a custom validation command component.	
Visible	Controls whether the component will be displayed or not. Can also be set programmatically using the component's `setVisible(boolean)` method.	

CHAPTER 3

Basic Tiled View

The Basic TiledView (tiled view) is a type of pagelet component. It is a special type of container view that can present its children (pagelets, and other visual components like display fields) in a number of repeated tiles, or repeated regions. Examples of tiles may be rows or columns of a table, or tabs in a tabbed component. There is no assumption of table layout made; simply the notion of repetition of tiles.

Property Name	Description	Notes
Auto Retrieving Models	A list of *retrieving type* models that will be have their `retrieve(...)` method invoked when the *retrieve WebAction* is invoked for this container view. The listed models must implement the `com.iplanet.jato.model.RetrievingModel` interface.	
Max Display Tiles	Controls how many tiles will be displayed. The default, -1, means unlimited tiles (display all that are available).	
Name	The class name of the component.	Req
Primary Model Dataset Name	Some models can have multiple parallel datasets (like Web service models). The primary dataset is the dataset that will be displayed by default when none is otherwise programmatically specified.	
Primary Model Reference	A tiled view can be associated with multiple models. The *primary* model controls the iterative tile behavior of the tiled view.	
Validation Exception Handler	Specifies how a validation exception should be handled: by the `handleValidationException(CommandEvent)` event, or with a custom validation command component.	
Visible	Controls whether the component will be displayed or not. Can also be set programmatically using the component's `setVisible(boolean)` method.	

CHAPTER 4

Basic Tree View

The Basic TreeView (tree view) is a type of pagelet component. It helps present information that is structured in a tree format, like XML and LDAP data structures.

Property Name	Description	Notes
Name	The class name of the component.	Req
Node Handle Request Handler	Specifies how an *expand tree node* request is handled (on the server side, not the client). The default is to expand the node and reload the page. Custom behavior can be defined in the tree view component's `handleTreeNodeHandleRequest(CommandEvent)` event, or in a custom command component.	
Primary Model Reference	A tree view can be associated with multiple models. The *primary* model controls the iterative node behavior of the tree view.	
State Data Session Attribute	The name of the HTTP session attribute used to store the state of the tree.	
Validation Exception Handler	Specifies how a validation exception should be handled: by the `handleValidationException(CommandEvent)` event, or with a custom validation command component.	
Visible	Controls whether the component will be displayed or not. Can also be set programmatically using the component's `setVisible(boolean)` method.	

CHAPTER 5

Basic ViewBean (Page)

The Basic ViewBean is also referred to as a page component. It is a special view component that functions as a root view in an arbitrarily complex view hierarchy. In other words, a ViewBean is a top-level view component that can contain other view components, but itself has no parent. ViewBeans can be thought of primarily as pages in your application. Basic ViewBeans (or 3rd party components which implement the `com.iplanet.jato.view.ViewBean` interface) are the only view components which can be executed (test run) from within the IDE.

Property Name	Description	Notes
Auto Deleting Models	A list of *deleting type* models that will be have their `delete(...)` method invoked when the *delete WebAction* is invoked for this container view. The listed models must implement the `com.iplanet.jato.model.DeletingModel` interface.	
Auto Executing Models	A list of *executing type* models that will be have their `execute(...)` method invoked when the *execute WebAction* is invoked for this container view. The listed models must implement the `com.iplanet.jato.model.ExecutingModel` interface.	
Auto Inserting Models	A list of inserting *type* models that will be have their `insert(...)` method invoked when the *insert WebAction* is invoked for this container view. The listed models must implement the `com.iplanet.jato.model.InsertingModel` interface.	
Auto Retrieving Models	A list of *retrieving type* models that will be have their `retrieve(...)` method invoked when the *retrieve WebAction* is invoked for this container view. The listed models must implement the `com.iplanet.jato.model.RetrievingModel` interface.	

Property Name	Description	Notes
Auto Updating Models	A list of updating *type* models that will be have their `update(...)` method invoked when the *update WebAction* is invoked for this container view. The listed models must implement the `com.iplanet.jato.model.UpdatingModel` interface.	
Name	The class name of the component.	Req
Validation Exception Handler	Specifies how a validation exception should be handled: by the `handleValidationException(CommandEvent)` event, or with a custom validation command component.	

CHAPTER 6

Button

A Button is a type of command field that submits form data. The button's request handling behavior is implemented in an instance-specific request handler method (`handle<`*ComponentName*`>Request`) or delegated to a command component.

Property Name	Description	Notes
Initial Value	The value to which the visual component is initialized upon its instantiation. Note, this value will overwrite any value in the bound model field if one exists. If you want to set a value on a component without potentially overwriting the model field's value, avoid using this property and instead use the `setValue(Object value,boolean overwrite)` method with the `overwrite` parameter set to `false`. You may call this method from your code as needed (for example, from an event handler) or from the Post-initialization Code property of the component.	
Model Field Binding	The model field to which the visual component is bound (where it stores/retrieves its value). The *Model Reference* property must be configured before this property can be configured.	Dependent on Model Reference
Model Reference	A reference to the model to which the visual component's bound model field belongs. This property must be configured before the *Model Field Binding* property can be configured.	Req
Name	The name of the component instance.	
Request Handler	Determines what will be the request handling mechanism for the command field. This could be the request event handler method (`handle<`*ComponentName*`>Request`), a custom command component, or a built-in command component, like a WebAction command, for example). The default setting is the request event handler method.	
Visible	Controls whether the component will be displayed or not. Can also be set programmatically using the component's `setVisible(boolean)` method.	

CHAPTER 7

Check Box

The Check Box component provides a mutually exclusive, two-state field value: true/false, yes/no, on/off, A/B, etc. The actual field types and values of the true and false states are completely customizable.

Property Name	Description	Notes
False Value	The type and value of the field when the visual component is in a false (unselected) state.	
Model Field Binding	The model field to which the visual component is bound (where it stores/retrieves its value). The *Model Reference* property must be configured before this property can be configured.	Dependent on Model Reference
Model Reference	A reference to the model to which the visual component's bound model field belongs. This property must be configured before the *Model Field Binding* property can be configured.	
Name	The name of the component instance.	Req
State	The initial state (value) of the check box (true or false).	
True Value	The type and value when the visual component is in a true (selected) state.	
Visible	Controls whether the component will be displayed or not. Can also be set programmatically using the component's `setVisible(boolean)` method.	

CHAPTER **8**

Combo Box

The Combo Box component is a type of choice component that provides a list of choices presented in a drop-down list style interface. The actual field types and values of the component's choices are customizable by the developer.

Property Name	Description	Notes
Choices	An array of choices that can be selected by the user. Each choice provides a value and a label.	
Initial Value	The value to which the visual component is initialized upon its instantiation. Note, this value will overwrite any value in the bound model field if one exists. If you want to set a value on a component without potentially overwriting the model field's value, avoid using this property and instead use the `setValue(Object value,boolean overwrite)` method with the `overwrite` parameter set to `false`. You may call this method from your code as needed (for example, from an event handler) or from the Post-initialization Code property of the component.	
Model Field Binding	The model field to which the visual component is bound (where it stores/retrieves its value). The *Model Reference* property must be configured before this property can be configured.	Dependent on Model Reference
Model Reference	A reference to the model to which the visual component's bound model field belongs. This property must be configured before the *Model Field Binding* property can be configured.	
Name	The name of the component instance.	Req
Null Choice Label	The text that is displayed as the component's choice when the field's value is null. Leaving this property empty (null) will prevent a null choice option from being presented to the user, thus forcing the user to select one of the choices from the list.	
Visible	Controls whether the component will be displayed or not. Can also be set programmatically using the component's `setVisible(boolean)` method.	

CHAPTER 9

Data-Driven Combo Box

The Data-Driven Combo Box component is a type of choice component that can have its choices populated from a model component. It provides a list of choices presented in a drop-down list style interface.

Property Name	Description	Notes
Cached Choices Attribute Name	The name of the attribute used to cache choices for reuse. This value is used in accordance with the value of the *Choices Retrieval Policy* property.	
Choices Label Binding	The model fields that will be used to generate the labels for the component's list of choices. The *Choices Model Reference* property must be configured before this property can be configured. This property allows multiple bindings, which will be combined via the *Choices Label Message Format* property into a single choice label.	Req Dependent on Choices Model Reference
Choices Label Message Format	The message format string that will be applied to transform the raw choice data into formatted choice labels. This format string may include plain text as well as standard Java message format tokens ("{0}", "{1}", etc.). Each message format token will be replaced by the value of the label model binding that corresponds to the specified index.	
Choices Model Reference	A reference to the model from which the component's choices list will be obtained using the value and choice label model field bindings. This property must be configured before the *Choices Label Binding* and *Choices Value Binding* properties can be configured.	Req
Choices Retrieval Exception Handler	Determines how an exception is handled if an exception is thrown while the choices values or labels are being retrieved from the model. The possible settings for this property are the following: the default (throw the Exception); invoke the data-driven choice component retrieval exception handler method (`handle`<*ComponentName*>`ChoicesRetrievalException`), or delegate to a command component.	

Property Name	Description	Notes
Choices Retrieval Policy	Determines how and when the choices for the component are retrieved and populated. The possible settings for this property are the following: manual retrieval (developer takes full control of initiating choices retrieval), once per request, once per session (once for each new HTTP session created), once per application (upon first request per virtual machine), and every time (if component is used more than once in a request, choices are retrieved once each time the component is rendered). Default setting is once per request.	
Choices Value Binding	The model fields that will be used to generate the values for the component's list of choices. The *Choices Model Reference* property must be configured before this property can be configured.	Req Dependent on Choices Model Reference
Initial Value	The value to which the visual component is initialized upon its instantiation. Note, this value will overwrite any value in the bound model field if one exists. If you want to set a value on a component without potentially overwriting the model field's value, avoid using this property and instead use the `setValue(Object value,boolean overwrite)` method with the `overwrite` parameter set to `false`. You may call this method from your code as needed (for example, from an event handler) or from the Post-initialization Code property of the component.	
Model Field Binding	The model field to which the visual component is bound (where it stores/retrieves its value). The *Model Reference* property must be configured before this property can be configured.	Dependent on Model Reference
Model Reference	A reference to the model to which the visual component's bound model field belongs. This property must be configured before the *Model Field Binding* property can be configured.	Req
Name	The name of the component instance.	
Null Choice Label	The text that is displayed as the component's choice when the field's value is null. Leaving this property empty (null) will prevent a null choice option from being presented to the user, thus forcing the user to select one of the choices from the list.	
Visible	Controls whether the component will be displayed or not. Can also be set programmatically using the component's `setVisible(boolean)` method.	

CHAPTER 10

Data-Driven List Box

The Data-Driven List Box component is a type of choice component that can have its choices populated from a model component. It presents its list of choices in a list box. Multiple choices, if allowed, can be selected by the user.

Property Name	Description	Notes
Allow Multiple Choices	Determines whether the user is allowed to select multiple choices or not. Default setting is `false`.	
Cached Choices Attribute Name	The name of the attribute used to cache choices for reuse. This value is used in accordance with the value of the *Choices Retrieval Policy* property.	
Choices Label Binding	The model fields that will be used to generate the labels for the component's list of choices. The *Choices Model Reference* property must be configured before this property can be configured. This property allows multiple bindings, which will be combined via the *Choices Label Message Format* property into a single choice label.	Req Dependent on Choices Model Reference
Choices Label Message Format	The message format string that will be applied to transform the raw choice data into formatted choice labels. This format string may include plain text as well as standard Java message format tokens ("{0}", "{1}", etc.). Each message format token will be replaced by the value of the label model binding that corresponds to the specified index.	
Choices Model Reference	A reference to the model from which the component's choices list will be obtained using the value and choice label model field bindings. This property must be configured before the *Choices Label Binding* and *Choices Value Binding* properties can be configured.	Req

Property Name	Description	Notes
Choices Retrieval Exception Handler	Determines how an exception is handled if an exception is thrown while the choices values or labels are being retrieved from the model. The possible settings for this property are the following: the default (throw the Exception); invoke the data-driven choice component retrieval exception handler method (`handle<`*ComponentName*`>ChoicesRetrievalException`), or delegate to a command component.	
Choices Retrieval Policy	Determines how and when the choices for the component are retrieved and populated. The possible settings for this property are the following: manual retrieval (developer takes full control of initiating choices retrieval), once per request, once per session (once for each new HTTP session created), once per application (upon first request per virtual machine), and every time (if component is used more than once in a request, choices are retrieved once each time the component is rendered). Default setting is once per request.	
Choices Value Binding	The model fields that will be used to generate the values for the component's list of choices. The *Choices Model Reference* property must be configured before this property can be configured.	Req Dependent on Choices Model Reference
Initial Value	The value to which the visual component is initialized upon its instantiation. Note, this value will overwrite any value in the bound model field if one exists. If you want to set a value on a component without potentially overwriting the model field's value, avoid using this property and instead use the `setValue(Object value,boolean overwrite)` method with the `overwrite` parameter set to `false`. You may call this method from your code as needed (for example, from an event handler) or from the Post-initialization Code property of the component.	
Model Field Binding	The model field to which the visual component is bound (where it stores/retrieves its value). The *Model Reference* property must be configured before this property can be configured.	Dependent on Model Reference
Model Reference	A reference to the model to which the visual component's bound model field belongs. This property must be configured before the *Model Field Binding* property can be configured.	
Name	The name of the component instance.	Req
Null Choice Label	The text that is displayed as the component's choice when the field's value is null. Leaving this property empty (null) will prevent a null choice option from being presented to the user, thus forcing the user to select one of the choices from the list.	
Visible	Controls whether the component will be displayed or not. Can also be set programmatically using the component's `setVisible(boolean)` method.	

CHAPTER 11

Data-Driven Radio Buttons

The Data-Driven Radio Buttons component is a type of choice component that can have its choices populated from a model component. It presents its list of choices as a group of radio buttons.

Property Name	Description	Notes
Cached Choices Attribute Name	The name of the attribute used to cache choices for reuse. This value is used in accordance with the value of the *Choices Retrieval Policy* property.	
Choices Label Binding	The model fields that will be used to generate the labels for the component's list of choices. The *Choices Model Reference* property must be configured before this property can be configured. This property allows multiple bindings, which will be combined via the *Choices Label Message Format* property into a single choice label.	Req Dependent on Choices Model Reference
Choices Label Message Format	The message format string that will be applied to transform the raw choice data into formatted choice labels. This format string may include plain text as well as standard Java message format tokens ("{0}", "{1}", etc.). Each message format token will be replaced by the value of the label model binding that corresponds to the specified index.	
Choices Model Reference	A reference to the model from which the component's choices list will be obtained using the value and choice label model field bindings. This property must be configured before the *Choices Label Binding* and *Choices Value Binding* properties can be configured.	Req
Choices Retrieval Exception Handler	Determines how an exception is handled if an exception is thrown while the choices values or labels are being retrieved from the model. The possible settings for this property are the following: the default (throw the Exception); invoke the data-driven choice component retrieval exception handler method (`handle`<*ComponentName*>`ChoicesRetrievalException`), or delegate to a command component.	

Property Name	Description	Notes
Choices Retrieval Policy	Determines how and when the choices for the component are retrieved and populated. The possible settings for this property are the following: manual retrieval (developer takes full control of initiating choices retrieval), once per request, once per session (once for each new HTTP session created), once per application (upon first request per virtual machine), and every time (if component is used more than once in a request, choices are retrieved once each time the component is rendered). Default setting is once per request.	
Choices Value Binding	The model fields that will be used to generate the values for the component's list of choices. The *Choices Model Reference* property must be configured before this property can be configured.	Req Dependent on Choices Model Reference
Initial Value	The value to which the visual component is initialized upon its instantiation. Note, this value will overwrite any value in the bound model field if one exists. If you want to set a value on a component without potentially overwriting the model field's value, avoid using this property and instead use the `setValue(Object value,boolean overwrite)` method with the `overwrite` parameter set to `false`. You may call this method from your code as needed (for example, from an event handler) or from the Post-initialization Code property of the component.	
Model Field Binding	The model field to which the visual component is bound (where it stores/retrieves its value). The *Model Reference* property must be configured before this property can be configured.	Dependent on Model Reference
Model Reference	A reference to the model to which the visual component's bound model field belongs. This property must be configured before the *Model Field Binding* property can be configured.	
Name	The name of the component instance.	Req
Null Choice Label	The text that is displayed as the component's choice when the field's value is null. Leaving this property empty (null) will prevent a null choice option from being presented to the user, thus forcing the user to select one of the choices from the list.	
Visible	Controls whether the component will be displayed or not. Can also be set programmatically using the component's `setVisible(boolean)` method.	

CHAPTER 12

File Upload

The File Upload component provides a way for users to send files to the server, and an easy way for the developer to gain access to the uploaded file content. Global application properties governing file upload can be configured in the application's Settings & Configuration node.

Property Name	Description	Notes
Name	The name of the component instance.	Req
Visible	Controls whether the component will be displayed or not. Can also be set programmatically using the component's `setVisible(boolean)` method.	

CHAPTER 13

Hidden Field

The Hidden Field component provides a way to embed non-visible data in a page so that it is sent back to the server when the surrounding form is posted.

Property Name	Description	Notes
Initial Value	The value to which the visual component is initialized upon its instantiation. Note, this value will overwrite any value in the bound model field if one exists. If you want to set a value on a component without potentially overwriting the model field's value, avoid using this property and instead use the `setValue(Object value,boolean overwrite)` method with the `overwrite` parameter set to `false`. You may call this method from your code as needed (for example, from an event handler) or from the Post-initialization Code property of the component.	
Model Field Binding	The model field to which the visual component is bound (where it stores/retrieves its value). The *Model Reference* property must be configured before this property can be configured.	Dependent on Model Reference
Model Reference	A reference to the model to which the visual component's bound model field belongs. This property must be configured before the *Model Field Binding* property can be configured.	
Name	The name of the component instance.	Req
Visible	Controls whether the component will be displayed or not. Can also be set programmatically using the component's `setVisible(boolean)` method.	

CHAPTER 14

Hyperlink (HREF)

The Hyperlink (HREF) component is a type of command field. Unlike the button, activation of a hyperlink does not cause form data to be submitted to the server. Instead, each hyperlink has its own set of query parameters. The hyperlink's request handling behavior is implemented in an instance-specific request handler method (`handle`<*ComponentName*>`Request`) or delegated to a command component.

Property Name	Description	Notes
Initial Value	The value to which the visual component is initialized upon its instantiation. Note, this value will overwrite any value in the bound model field if one exists. If you want to set a value on a component without potentially overwriting the model field's value, avoid using this property and instead use the `setValue(Object value,boolean overwrite)` method with the `overwrite` parameter set to `false`. You may call this method from your code as needed (for example, from an event handler) or from the Post-initialization Code property of the component.	
Model Field Binding	The model field to which the visual component is bound (where it stores/retrieves its value). The *Model Reference* property must be configured before this property can be configured.	Dependent on Model Reference
Model Reference	A reference to the model to which the visual component's bound model field belongs. This property must be configured before the *Model Field Binding* property can be configured.	
Name	The name of the component instance.	Req
Request Handler	Determines what will be the request handling mechanism for the command field. This could be the request event handler method (`handle`<*ComponentName*>`Request`), a custom command component, or a built-in command component, like a WebAction command, for example). The default setting is the request event handler method.	
Visible	Controls whether the component will be displayed or not. Can also be set programmatically using the component's `setVisible(boolean)` method.	

CHAPTER 15

Image

The Image component allows an image to be displayed on a page. This component should be used when the URL of the image is dynamically determined by the application (static images can simply be encoded in the page's markup).

Property Name	Description	Notes
Initial Value	The value to which the visual component is initialized upon its instantiation. Note, this value will overwrite any value in the bound model field if one exists. If you want to set a value on a component without potentially overwriting the model field's value, avoid using this property and instead use the `setValue(Object value,boolean overwrite)` method with the `overwrite` parameter set to `false`. You may call this method from your code as needed (for example, from an event handler) or from the Post-initialization Code property of the component.	
Model Field Binding	The model field to which the visual component is bound (where it stores/retrieves its value). The *Model Reference* property must be configured before this property can be configured.	Dependent on Model Reference
Model Reference	A reference to the model to which the visual component's bound model field belongs. This property must be configured before the *Model Field Binding* property can be configured.	
Name	The name of the component instance.	Req
Visible	Controls whether the component will be displayed or not. Can also be set programmatically using the component's `setVisible(boolean)` method.	

CHAPTER 16

List Box

The List Box component is a type of choice component that presents its list of choices in a list box. Multiple choices, if allowed, can be selected by the user.

Property Name	Description	Notes
Allow Multiple Choices	Determines whether the user is allowed to select multiple choices or not. Default setting is `false`.	
Choices	An array of choices that can be selected by the user. Each choice provides a value and a label.	
Initial Value	The value to which the visual component is initialized upon its instantiation. Note, this value will overwrite any value in the bound model field if one exists. If you want to set a value on a component without potentially overwriting the model field's value, avoid using this property and instead use the `setValue(Object value,boolean overwrite)` method with the `overwrite` parameter set to `false`. You may call this method from your code as needed (for example, from an event handler) or from the Post-initialization Code property of the component.	
Model Field Binding	The model field to which the visual component is bound (where it stores/retrieves its value). The *Model Reference* property must be configured before this property can be configured.	Dependent on Model Reference
Model Reference	A reference to the model to which the visual component's bound model field belongs. This property must be configured before the *Model Field Binding* property can be configured.	

Property Name	Description	Notes
Name	The name of the component instance.	Req
Null Choice Label	The text that is displayed as the component's choice when the field's value is null. Leaving this property empty (null) will prevent a null choice option from being presented to the user, thus forcing the user to select one of the choices from the list.	
Visible	Controls whether the component will be displayed or not. Can also be set programmatically using the component's `setVisible(boolean)` method.	

CHAPTER 17

Password Field

The Password Field component provides a way for the users to enter text without showing the characters they have entered. Instead, asterisk are shown for each character.

Property Name	Description	Notes
Initial Value	The value to which the visual component is initialized upon its instantiation. Note, this value will overwrite any value in the bound model field if one exists. If you want to set a value on a component without potentially overwriting the model field's value, avoid using this property and instead use the `setValue(Object value,boolean overwrite)` method with the `overwrite` parameter set to `false`. You may call this method from your code as needed (for example, from an event handler) or from the Post-initialization Code property of the component.	
Model Field Binding	The model field to which the visual component is bound (where it stores/retrieves its value). The *Model Reference* property must be configured before this property can be configured.	Dependent on Model Reference
Model Reference	A reference to the model to which the visual component's bound model field belongs. This property must be configured before the *Model Field Binding* property can be configured.	
Name	The name of the component instance.	Req
Visible	Controls whether the component will be displayed or not. Can also be set programmatically using the component's `setVisible(boolean)` method.	

CHAPTER 18

Radio Buttons

The Radio Buttons component is a type of choice component that presents its list of choices as a group of radio buttons. The choices presented by the component are developer-defined via the Choices property.

Property Name	Description	Notes
Choices	Choices An array of choices that can be selected by the user. Each choice provides a value and a label.	
Initial Value	The value to which the visual component is initialized upon its instantiation. Note, this value will overwrite any value in the bound model field if one exists. If you want to set a value on a component without potentially overwriting the model field's value, avoid using this property and instead use the `setValue(Object value,boolean overwrite)` method with the `overwrite` parameter set to `false`. You may call this method from your code as needed (for example, from an event handler) or from the Post-initialization Code property of the component.	
Model Field Binding	The model field to which the visual component is bound (where it stores/retrieves its value). The *Model Reference* property must be configured before this property can be configured.	Dependent on Model Reference
Model Reference	A reference to the model to which the visual component's bound model field belongs. This property must be configured before the *Model Field Binding* property can be configured.	
Name	The name of the component instance.	Req
Null Choice Label	The text that is displayed as the component's choice when the field's value is null. Leaving this property empty (null) will prevent a null choice option from being presented to the user, thus forcing the user to select one of the choices from the list.	
Visible	Controls whether the component will be displayed or not. Can also be set programmatically using the component's `setVisible(boolean)` method.	

CHAPTER 19

Static Text Field

The Static Text Field component displays read-only text or markup. This component can be used to display user-visible text (for example, labels on a page), or used to generate markup or other non-visual text content.

Property Name	Description	Notes
Initial Value	The value to which the visual component is initialized upon its instantiation. Note, this value will overwrite any value in the bound model field if one exists. If you want to set a value on a component without potentially overwriting the model field's value, avoid using this property and instead use the `setValue(Object value,boolean overwrite)` method with the `overwrite` parameter set to `false`. You may call this method from your code as needed (for example, from an event handler) or from the Post-initialization Code property of the component.	
Model Field Binding	The model field to which the visual component is bound (where it stores/retrieves its value). The *Model Reference* property must be configured before this property can be configured.	Dependent on Model Reference
Model Reference	A reference to the model to which the visual component's bound model field belongs. This property must be configured before the *Model Field Binding* property can be configured.	
Name	The name of the component instance.	Req
Visible	Controls whether the component will be displayed or not. Can also be set programmatically using the component's `setVisible(boolean)` method.	

CHAPTER 20

Text Field

The Text Field component is a single-line, free-form text input field.

Property Name	Description	Notes
Initial Value	The value to which the visual component is initialized upon its instantiation. Note, this value will overwrite any value in the bound model field if one exists. If you want to set a value on a component without potentially overwriting the model field's value, avoid using this property and instead use the `setValue(Object value,boolean overwrite)` method with the `overwrite` parameter set to `false`. You may call this method from your code as needed (for example, from an event handler) or from the Post-initialization Code property of the component.	
Model Field Bindings	The model field to which the visual component is bound (where it stores/retrieves its value). The *Model Reference* property must be configured before this property can be configured.	Dependent on Model Reference
Model Reference	A reference to the model to which the visual component's bound model field belongs. This property must be configured before the *Model Field Binding* property can be configured.	
Name	The name of the component instance.	Req
Visible	Controls whether the component will be displayed or not. Can also be set programmatically using the component's `setVisible(boolean)` method.	

CHAPTER 21

Text Area

The Text Area component is a multi-line, free-form text input field.

Property Name	Description	Notes
Initial Value	The value to which the visual component is initialized upon its instantiation. Note, this value will overwrite any value in the bound model field if one exists. If you want to set a value on a component without potentially overwriting the model field's value, avoid using this property and instead use the `setValue(Object value,boolean overwrite)` method with the `overwrite` parameter set to `false`. You may call this method from your code as needed (for example, from an event handler) or from the Post-initialization Code property of the component.	
Model Field Binding	The model field to which the visual component is bound (where it stores/retrieves its value). The *Model Reference* property must be configured before this property can be configured.	Dependent on Model Reference
Model Reference	A reference to the model to which the visual component's bound model field belongs. This property must be configured before the *Model Field Binding* property can be configured.	Req
Name	The name of the component instance.	
Visible	Controls whether the component will be displayed or not. Can also be set programmatically using the component's `setVisible(boolean)` method.	

CHAPTER 22

Validating Text Field

The Validating Text Field component is a single-line, free-form text input field with the ability to validate the user-supplied text using an associated validation component.

Property Name	Description	Notes
Initial Value	The value to which the visual component is initialized upon its instantiation. Note, this value will overwrite any value in the bound model field if one exists. If you want to set a value on a component without potentially overwriting the model field's value, avoid using this property and instead use the `setValue(Object value,boolean overwrite)` method with the `overwrite` parameter set to `false`. You may call this method from your code as needed (for example, from an event handler) or from the Post-initialization Code property of the component.	
Model Field Binding	The model field to which the visual component is bound (where it stores/retrieves its value). The *Model Reference* property must be configured before this property can be configured.	Dependent on Model Reference
Model Reference	A reference to the model to which the visual component's bound model field belongs. This property must be configured before the *Model Field Binding* property can be configured.	
Name	The name of the component instance.	Req
Show Validation Failure Message	Determines whether to show the value of the *Validation Failure Message* property when validation fails for the component.	

Property Name	Description	Notes
Validation Failure Message	The text message to optionally display to the end user when validation fails for the field. This value may contain markup.	
Validator	The validation component associated with the field.	
Visible	Controls whether the component will be displayed or not. Can also be set programmatically using the component's `setVisible(boolean)` method.	

CHAPTER 23

Validating Text Area

The Validating Text Area component is a multi-line, free-form text input field with the ability to validate the user-supplied text using an associated validation component.

Property Name	Description	Notes
Initial Value	The value to which the visual component is initialized upon its instantiation. Note, this value will overwrite any value in the bound model field if one exists. If you want to set a value on a component without potentially overwriting the model field's value, avoid using this property and instead use the `setValue(Object value,boolean overwrite)` method with the `overwrite` parameter set to `false`. You may call this method from your code as needed (for example, from an event handler) or from the Post-initialization Code property of the component.	
Model Field Binding	The model field to which the visual component is bound (where it stores/retrieves its value). The *Model Reference* property must be configured before this property can be configured.	Dependent on Model Reference
Model Reference	A reference to the model to which the visual component's bound model field belongs. This property must be configured before the *Model Field Binding* property can be configured.	
Name	The name of the component instance.	Req
Show Validation Failure Message	Determines whether to show the value of the *Validation Failure Message* property when validation fails for the component.	

Property Name	Description	Notes
Validation Failure Message	The text message to optionally display to the end user when validation fails for the field. This value may contain markup.	
Validator	The validation component associated with the field.	
Visible	Controls whether the component will be displayed or not. Can also be set programmatically using the component's `setVisible(boolean)` method.	

CHAPTER 24

Masked Text Field

The Masked Text Field component is a single-line, free-form text input field. This component adds support for the validation of input against a specified mask expression.

A mask expression is specified as:

- a # character in the mask signifies a number (0-9)
- a A character in the mask signifies an upper case letter
- a a (small a) character in the mask signifies a lower case letter
- a B character in the mask signifies a letter (upper or lower case)
- a . character in the mask signifies any character
- a * character in the mask signifies any letter or number
- any other character (except the \) is required as is
- a \ character precedes any character that requires escaping (only a #, A, *, . and \)

For example, a mask expression of "###-##-####" could be used to validate a social security number input. A mask expression of "\##B#\#" describes input that begins and ends with "#" with a letter in between two numbers in the middle (ex. "#9k1#")

Note – The backslash character "\" used for escaping characters will require additional escaping to compensate for the Java and Javascript code used in this component. For example, an input mask of "\#aA*" will require an input of "\\\\ #aA*" in the component.

If you do not use an input mask expression, this component behaves much like a standard Text Field.

Property Name	Description	Notes
Initial Value	The value to which the visual component is initialized upon its instantiation. Note, this value will overwrite any value in the bound model field if one exists. If you want to set a value on a component without potentially overwriting the model field's value, avoid using this property and instead use the `setValue(Object value,boolean overwrite)` method with the `overwrite` parameter set to `false`. You may call this method from your code as needed (for example, from an event handler) or from the Post-initialization Code property of the component.	
Model Field Binding	The model field to which the visual component is bound (where it stores/retrieves its value). The *Model Reference* property must be configured before this property can be configured.	Dependent on Model Reference
Model Reference	A reference to the model to which the visual component's bound model field belongs. This property must be configured before the *Model Field Binding* property can be configured.	
Name	The name of the component instance.	Req
Visible	Controls whether the component will be displayed or not. Can also be set programmatically using the component's `setVisible(boolean)` method.	
Mask Expression	Masked expression value in which input of this component is validated against.	

CHAPTER 25

Date View

The Date View component is a visual display field component which displays the date whose fields (month, day, year) are represented by Combo Boxes. Also includes a javascript mini-calendar popup.

The Date View component is available from the component palette and is added to a Container View is the same way as other components. The Date View component displays the date portion of the value of its model field binding. A javascript mini-calendar popup can be configured with this component through the Show Mini-Calendar property. Two other configuration properties (Max/Min Year Choice Display Value) can be set to limit the size of the year field combo box.

Property Name	Description	Notes
Model Reference	A reference to the model to which the visual component's bound model field belongs. This property must be configured before the *Model Field Binding* property can be configured.	Req
Model Field Binding	The model field to which the visual component is bound (where it stores/retrieves its value). The *Model Reference* property must be configured before this property can be configured.	Dependent on Model Reference
Name	The name of the component instance.	
Max Year Choice Display Value	Specify the maximum year to display for the year field combo box.	Default is <current year> + 50
Min Year Choice Display Value	Specify the minimum year to display for the year field combo box.	Default is <current year> - 50
Show Mini-Calendar	Specify whether to display the button to invoke the mini-calendar popup.	Default is false.

CHAPTER 26

Time View

The Time View component is a visual display field component which displays the time whose fields (hour, minute, etc.) are represented by Combo Boxes.

The Time View component is available from the component palette and is added to a Container View is the same way as other components. The Time View component displays the time portion of the value of its model field binding. In the military time display format, two combo boxes are used to represent the hour and minute fields. In the standard time display format, three combo boxes are used to represent the hour, minute, and am/pm fields.

Property Name	Description	Notes
Model Reference	A reference to the model to which the visual component's bound model field belongs. This property must be configured before the *Model Field Binding* property can be configured.	Req
Model Field Binding	The model field to which the visual component is bound (where it stores/retrieves its value). The *Model Reference* property must be configured before this property can be configured.	Dependent on Model Reference
Name	The name of the component instance.	
Visible	Controls whether the component will be displayed or not. Can also be set programmatically using the component's `setVisible(boolean)` method.	
Minute Interval Display	Specify the minute display increments in intervals of 1, 5, 10, 15, or 30 minutes.	Default is 1.
Time Format	Specify whether the display format is in military time (24 hour clock) or standard time (12 hour clock).	Default is 12 hour.

CHAPTER 27

DateTime View

The DateTime View component is a visual display field component which displays the date whose fields (month, day, year, hour, minute, etc.) are represented by Combo Boxes. Also includes a javascript mini-calendar popup.

The DateTime View component is available from the component palette and is added to a Container View is the same way as other components. The DateTime View component displays the date and time portion of the value of its model field binding. A javascript mini-calendar popup can be configured with this component through the Show Mini-Calendar property. Two other configuration properties (Max/Min Year Choice Display Value) can be set to limit the size of the year field combo box. In the military time display format, two combo boxes are used to represent the hour and minute fields. In the standard time display format, three combo boxes are used to represent the hour, minute, and am/pm fields.

Property Name	Description	Notes
Model Reference	A reference to the model to which the visual component's bound model field belongs. This property must be configured before the *Model Field Binding* property can be configured.	Req
Model Field Binding	The model field to which the visual component is bound (where it stores/retrieves its value). The *Model Reference* property must be configured before this property can be configured.	Dependent on Model Reference
Name	The name of the component instance.	
Max Year Choice Display Value	Specify the maximum year to display for the year field combo box.	Default is <current year> + 50
Min Year Choice Display Value	Specify the minimum year to display for the year field combo box.	Default is <current year> - 50
Show Mini-Calendar	Specify whether to display the button to invoke the mini-calendar popup.	Default is false.

Property Name	Description	Notes
Minute Interval Display	Specify the minute display increments in intervals of 1, 5, 10, 15, or 30 minutes.	Default is 1
Time Format	Specify whether the display format is in military time (24 hour clock) or standard time (12 hour clock).	Default is 12 hour

CHAPTER 28

Go To Page Link

The Go To Page Link component is a HREF display field component which is preconfigured to use a Goto View Bean command descriptor.

The Go To Page Link component behaves in much the same way as the Hyperlink (HREF) component. The only difference is that the Go To Page Link component is explicitly preconfigured with a Goto View Bean command descriptor. This descriptor takes its parameter from the Target ViewBean Class Name property of the this component.

Property Name	Description	Notes
Model Reference	A reference to the model to which the visual component's bound model field belongs. This property must be configured before the *Model Field Binding* property can be configured.	Req
Model Field Binding	The model field to which the visual component is bound (where it stores/retrieves its value). The *Model Reference* property must be configured before this property can be configured.	Dependent on Model Reference
Name	The name of the component instance.	
Visible	Controls whether the component will be displayed or not. Can also be set programmatically using the component's setVisible(boolean) method.	
Target ViewBean Class Name	The class name of the view bean that the will be displayed at the conclusion of the request.	Req

CHAPTER 29

Menu

Menu is a View component that allows user to design drop-down menu functionality into their web applications. Menu component consists of Menu bar and sub menu items. Menu component is available from the component palette and one can configure the style and look of the menu at design time.

Hierarchical collection of menu items can be configured by defining the menu items through XML. At run time, this XML will be used as an input for reading the menu definition.

For more details on writing this XML, please refer to the DTD and sample XML.

Property Name	Description	Notes
Background Color	This property defines the background color of the menu. All menu items will be shown with the color that is specified by this property.	
Mouse over color	This property defines the mouse over color of the menu. When the mouse is moved over the menu items, they will change to the color specified by this property.	
Font Name	This property defines the font family to be used for the menu item(s) text. All menu item text will appear with the font that is specified by this property.	
Font Size	This property defines the font size to be used for the menu item(s) text. All menu item text will appear with the font size that is specified by this property.	
Font Style	This property defines the font style to be used for the menu item(s) text. All menu item text will appear with the font style that is specified by this property.	

Property Name	Description	Notes
Menu Style	This property defines the style of the menu. (Either HORIZONTAL or VERTICAL)	
Menu Definition File	This property defines the name of the XML file which has the menu definition. The location of the XML file should be relative to the context root of the application.	Req
Cached Menu Definition Attribute Name	This property defines the application scope attribute name where the XML DOM Tree is kept. When this is set, menu definition will be read from the DOM tree stored in the attribute specified by this property.	

CHAPTER 30

Static Breadcrumb

Breadcrumb navigation displays the current page's context within the site structure. It also helps user getting to know the ways in which information has been grouped and allows him to move between these groupings and understand the information structure. Static Breadcrumbs are a type of Breadcrumb where in, a Breadcrumb displayed for the current page does not depend on the user navigation, but depends on the logical grouping of pages, specified by the site developer.

Static Breadcrumb is a View component, that allows the user to drag and drop static breadcrumb components into their web applications. The web application can be mapped to hierarchical collection of breadcrumb items, that can be configured though XML. At run time, this XML will be used as an input for reading the site map structure and depending on the page currently displayed, the bread crumb will be constructed.

For more details on writing this XML, please refer to the DTD and sample XML.

Property Name	Description	Notes
Prefix Text	This property defines the text that will be rendered pre-fixing the Breadcrumb. For example, If Prefix Text property is *You are currently here* it will be rendered as: You are currently here : Home > Shopping > Men's Section in the browser.	
Separator Image	This property defines the separator image to be used between breadcrumb items. Both relative and absolute URL can be specified. In case of relative URL, the location of the image file should be relative to the context root of the application.	If separator image is not specified, then Separator String will be used as a separator.
Separator String	This property defines the separator string to be used between breadcrumb items. For example, If Separator String property is " >> " it will be rendered as You are currently here : Home >> Shopping >> Men's Section in the browser.	The string specified in this property will also be used as an alternate separator when the image can not be rendered by the browser.
Site Map File	This property defines the name of the XML file which has the site map structure. The location of the XML file should be relative to the context root of the application.	
Cached Site Map Attribute Name	This property defines the application scope attribute name where the XML DOM Tree is kept. When this is set, site map definition will be read from the DOM tree stored in the attribute specified by this property.	

CHAPTER 31

Dataset Navigator

The Dataset Navigator component is a set of four command fields that enable pagination (navigation) control over a set of data through a container view, like a BasicViewBean, a BasicTiledView or a BasicContainerView. Each of the four command fields represents the four types of navigation: First, Previous, Next, and Last. The command fields can be configured to auto-hide or auto-disable when the operation would result in no change to the display. For example, if the last record is currently displayed, the Next and Last command fields would be hidden or disabled since there is no *next* record.

Property Name	Description	Notes
Auto Disable	Enables the command fields to auto hide/disable when the corresponding operation would not result in a difference in the displayed data.	
Hide When Disabled	Prevents the display of the command fields that are in a disabled mode.	Dependent on Auto Disable
Name	The class name of the component.	

Target Container View Path	The name of the ContainerView reference instance as declared in its parent (a ContainerView, TiledView, or ViewBean).
	This name may be a qualified view path, using forward slashes ("/") as delimiters. All components in the path must refer to a ContainerView or a derivative of ContainerView (such as TiledView, ContainerView, or ViewBean or other custom or third party version derivative of these types). Both relative and absolute paths are possible. If a name path begins with a forward slash, the name is assumed to be relative to the root view (the ViewBean). If the path does not begin with a forward slash, the name is assumed to refer to a child relative to the current container. Two dots ("..") may be used to refer to the container that is the parent of the current container.
	Examples:
	`/header/orderList/customerName (absolute from root view)`
	`orderList/customerName (relative to current container)`
	`../footer/orderList/customerName (relative to parent)`	

CHAPTER 32

Dataset Locator

The Dataset Locator component displays the *record* of the displayed data (e.g. Records 1 to 10 of 53).

The “to ##” portion will only be displayed if the page displays more than one record at a time.

The “of ##” portion will only be displayed if it can be calculated. Some models do not fully implement the `PaginatingModel` interface, in particular, the `getTotalDataCount` method, and therefore, this information is not available. Some models read the entire dataset into memory on each request, and therefore, the total is always available, even if the `getTotalDataCount` method is not implemented.

If a page is displayed with no records, the DatasetLocator will display “(No Data Found)” by default. This message can be customized by editing the JSP pagelet, “com/sun/jatox/view/DatasetLocator.jsp”. This will happen in the event that a model does not implement the `getTotalDataCount` method, and the actual total data count is evenly divisible by the number of records displayed per page. On the last page of data of 50 total records with a 10 record per page display, the display will read “Records 41 to 50”, and because the total data count is unknown, the Next and Last buttons will still be enabled. When the user clicks one of these buttons, a blank page will be displayed with the "(No Data Found)" display and the Next and Last buttons will disabled at this point.

Physcial layout of the DatasetLocator component on the page may determine whether it works or not. This is because of the sequence in which content is generated versus the actual execution of models in the application. For example, if you place the DatasetLocator before the fields that actually display the data, the model may not have been executed yet. If it has not, either move the DatasetLocator to follow the data display, or just execute the model manually before the DatasetLocator is displayed.

Property Name	Description	Notes
Show Dataset Total	Displays the absolute total number of records in the dataset (last record index number). This will only work for model types that have properly implemented the `getTotalDataSize` method from the `PaginatingModel` interface, or the total number of records can be deterministally calculated. If the total number of records can not be determined, then the total records portion of this control will not display, as if the property were set to false.	
Name	The class name of the component.	

Target Container View Path	The name of the ContainerView reference instance as declared in its parent (a ContainerView, TiledView, or ViewBean).
	This name may be a qualified view path, using forward slashes ("/") as delimiters. All components in the path must refer to a ContainerView or a derivative of ContainerView (such as TiledView, ContainerView, or ViewBean or other custom or third party version derivative of these types). Both relative and absolute paths are possible. If a name path begins with a forward slash, the name is assumed to be relative to the root view (the ViewBean). If the path does not begin with a forward slash, the name is assumed to refer to a child relative to the current container. Two dots ("..") may be used to refer to the container that is the parent of the current container.
	Examples:
	`/ (target the same container view that is also the parent of this component)`
	`/header/orderList/customerName (absolute from root view)`
	`orderList/customerName (relative to current container)`
	`../footer/orderList/customerName (relative to parent)`	

CHAPTER 33

Bean Adapter Model

The Bean Adapter Model allows developers to use one or more JavaBeans as the backing datastore for a model. This allows display fields to be bound to JavaBean properties, and is a convenient approach when you have an application object model and want to leverage automatic binding of these objects to a view. This model is an ideal solution for integrating with an EJB client library. The common and recommended approach when designing a client EJB interface is to use the transfer object pattern where the input, output and return parameters of the EJB business methods are primitive or JavaBeans or collections of the same.

Property Name	Description	Notes
Bean Class	The fully qualified class name of the JavaBean that the model is *adapting* (based on).	
Bean Scope	The J2EE scope or location of the bean to be adapted: request, session, application, none or any are the choices. Please note that this model will not originate or create the bean. Existence of the bean in whatever scope is not a responsibility of the base class implementation. The developer may choose none and programmatically assign the adapted bean(s).	
Bean Scope Attribute Name	The name of the attribute when the scope is set to request, session or application. For example, if the bean is session scoped, this property should be set to the name of the HTTP session attribute.	
Name	The class name of the component.	Req

Bean Adapter Model Design Actions

Update Fields

Provides for the automatic validation of the Bean Class property and the creation of a model field for each JavaBean property of the Bean Class. This mechanism uses JavaBean introspection to determine the properties. Since the Introspector may cache BeanInfo, repeated invocations of this design action will usually yield the same set of JavaBean properties. If the adapted Bean Class itself changes, then the Studio filesystem for that Bean Class will need to be remounted so that the Bean Introspector will pick up the latest property descriptors.

Fields

Property Name	Description	Notes
Bean Property Name	The name of JavaBean property for this field. Each model field on the Bean Adapter Model represents a single JavaBean property. This property value may be null (not set) if the developer wishes to rely on the model field name property to also represent the name of the JavaBean property. If this property value is set, it must match the exact name of the JavaBean property descriptor name.	
Name	The logical name of the model field.	Req

CHAPTER 34

Custom Model

Custom Model extensible components support developers who need to create completely new and arbitrary model implementations. While it has always been possible to code a new model implementation manually, using custom models, one may have the new model and its fields and operations exposed in the IDE, automatically. The custom model provides little existing infrastructure and no storage for model data. The custom model is a solution for the developer who would have previously coded a new model from scratch by minimally implementing the Model interface.

Property Name	Description	Notes
Default Operation Name	The operation which may be invoked when an operation name is not specified in the ModelExecutionContext parameter of the `execute()` method. Please note that the `execute()` method implementation of the new model must be provided by the developer. If the developer chooses to disable or omit `execute()` behavior for their new model then this property will have no purpose. In short, use of this property is at the discretion of the component author.	
Name	The class name of the component.	Req

Fields

Property Name	Description	Notes
Field Class	Class type of the field (String, Integer, Boolean, etc.) which may be leveraged by the developer coding the model implementation.	
Name	The logical name of the model field.	Req

Operations

Property Name	Description	Notes
Name	The model operation name.	Req

CHAPTER 35

Simple Custom Model

The Simple Custom Model provides a foundation for a new model which requires advanced dataset management and pagination support. The simple custom model is a solution for the developer who would have previously coded a new model which specialized com.iplanet.jato.model.DefaultModel. Unlike other types of custom model, this type provides field value storage and other behavior, allowing the developer to merely customize existing capabilities rather then define them.

Property Name	Description	Notes
Coerce Value Types	When true, the model implementation will try to convert values set on the model fields to the type specified for that field. For instance if the type of the field is Boolean, then a string value from an HTML form which is mapped to the model through a display field will be coerced to a Boolean. This feature uses the com.iplanet.jato.util.TypeConverter class for all type conversions, which allows developers to register new type conversion algorithms.	
Default Operation Name	The operation which may be invoked when an operation name is not specified in the ModelExecutionContext parameter of the `execute()` method. Please note that the `execute()` method implementation of the new model must be provided by the developer. If the developer chooses to disable or omit `execute()` behavior for their new model then this property will have no purpose. In short, use of this property is at the discretion of the component author.	
Name	The class name of the component.	Req

Fields

Property Name	Description	Notes
Field Class	Class type of the field (String, Integer, Boolean, etc.) which may be leveraged by the developer coding the model implementation.	
Name	The logical name of the model field.	Req

Operations

Property Name	Description	Notes
Name	The model operation name.	Req

CHAPTER 36

Custom Tree Model

The Custom Tree Model allows a developer to use data stores that use a hierarchical (tree or directory like) data structure, like XML documents, LDAP repositories, or file systems.

Property Name	Description	Notes
Name	The class name of the component.	Req

CHAPTER 37

HTTP Session Model

An HTTP Session Model uses the HTTP session as its backing data store. HTTP session model fields map directly to HTTP session attributes. Unlike most other models, the fields of an HTTP session model are merely a passthrough vehicle for the values to the session attributes. In other words, setting the value on a field pushes the value immediately into the session attribute without the need to execute the model to update the actual backing data store. This model has no internal storage it is a facade to the session attributes.

HTTP session model is not required for interaction with the HTTP Session within a Web Application Framework application. The HTTP session API is completely accessible as it is in any web application. This model provides an avenue for the developer to formally declare session attributes during application design time for use in binding and potentially as an adapter or interceptor to HttpSession.

Property Name	Description	Notes
Allow Setting Equivalent Value	In certain application servers, the affect of calling `HttpSession.setAttribute()` may have consequences, including invoking `HttpSessionBindingListeners` and or causing transactions on a persistent session store. This property provides a way for redundant `setValue()` calls to be defensive against passthrough to `HttpSession.setAttribute()`.	
Name	The class name of the component.	Req

Fields

Property Name	Description	Notes
Attribute Class	Class type of the HTTP session attribute. The HTTP Session Model implementation of setValue() will use this property to coerce value types.	
Attribute Name	The name of the HTTP session attribute to which the model field maps. This property value may be null (not set) if the developer wishes to rely on the model field name property to also represent the name of the HTTP session attribute. If this property value is set, it must match the exact name intended for the session attribute.	
Name	The logical name of the model field.	Req
Optional Initial Value	Allows an initial value to encapsulated with the field declaration. In order to have session attributes initialized based on this property, the developer should acquire this model from the ModelManager and invoke the `ensureInitialValues()` method to push all initial values into HttpSession. The proper place to make this call is from the application servlet `onNewSession()` method.	
Store As Transient Attribute	When set to `true`, calls to `setValue()` for this field will wrap the actual value in a JavaBean which manages the value as a transient member. The potential value here is when an application server supports the passivation or persistent storage of HttpSession, this property may be used to support an in memory cache of the attribute value. For instance, if developer has a large amount of business data which is needed across a user session, he or she can avoid having this large data structure pushed to disk or persistent store. This is an advanced property which should only be enabled with care. The result of enabling this property is that this session attribute will not be highly available in the case of an application server which provides high availability support for the HTTP session.	

CHAPTER 38

JDBC SQL Query Model

The JDBC SQL Query Model allows developers to use one or more RDBMS tables as the backing datastore for the model. This allows display fields to be bound to columns in the database tables. All of the SQL operations can be performed using the query model: select (including multi-table joins), insert, update, and delete.

Property Name	Description	Notes
Data Source	The JDBC datasource name that will be used to obtain a connection from the J2EE container.	Req
Modifying Query Table	The name of the table that will be used when generating modifying queries (insert, update, delete). The model may use several tables for the select query but may only use one for modifying queries.	
Name	The class name of the component.	Req
Select SQL Template	The SQL select statement template used to construct SQL SELECT statements for a retrieve operation. Typically, there is a where token ("__WHERE__") at the end of this statement that is replaced by a where clause that is constructed dynamically at runtime by the component.	
Static Where Criteria	The where clause that is used with *every* SQL operation (except insert).	

Fields

Property Name	Description	Notes
Column Name	The actual name of the column in the table to which the model field maps.	Req
Computed Field	True if If the model field is mapped to a computed field (a SQL aggregate function). Default setting is false.	
Empty Formula	Specifies a SQL formula that provides a value if the field has no value, and only if the *Empty Value Policy* property is set to Use *Formula*.	
Empty Value Policy	The policy used to provide a value for the field during an update operation. Choices are Exclude, Send Null, and Use Formula. Default setting is Exclude.	
Field Type	The Java class type of the model field (java.lang.String, java.lang.Integer, java.lang.Boolean, etc.).	Req
Insert Formula	Specifies a SQL formula that provides a value if the field has no value, and only if the *Insert Value Source* property is set to *Use Formula*.	
Insert Value Source	The policy used to provide a value for the field during an insert operation. Choices are Application, Database, and Use Formula. Default setting is Application.	
Key Field	Specifies that the column to which the model maps is a key field or not. This property is required to be set if this model will be used for update, insert or delete behavior. When creating a Query Model using the wizard, it is not always possible for the key fields to be found in the JDBC driver metadata. For instance, PointBase datasources often fail to reveal key field indications while Oracle datasources work very consistently. If this field is not set and the update, insert or delete behavior is invoked, it may lead to a SQL exception.	
Name	The logical name of the model field.	Req
Qualified Column Name	The fully qualified name (<*table*>.<*column*>) of the column to which the model field maps.	Req
Supported Operations	The SQL operations in which the model field will participate: Select, Insert, Update, and Delete. Default setting is Select, Insert, Update, and Delete.	

CHAPTER 39

JDBC Stored Procedure Model

The JDBC Stored Procedure model allows developers to execute stored procedures. This allows display fields to be bound to parameters and result columns (where vendor supported) in the stored procedure.

Property Name	Description	Notes
Data Source	The JDBC datasource name that will be used to obtain a connection from the J2EE container.	Req
Name	The class name of the component.	Req
Procedure Name	The name of the stored procedure in the RDBMS which this model will invoke.	Req

Result Set Column Fields

Property Name	Description	Notes
Column Name	The name of the result column to which the model field maps in the stored procedure.	Req
Field Type	The Java class type of the model field (java.lang.String, java.lang.Integer, java.lang.Boolean, etc.).	Req
Name	The logical name of the model field.	Req

Procedure Parameter Fields

Property Name	Description	Notes
Parameter Class	The Java class type of the model field (java.lang.String, java.lang.Integer, java.lang.Boolean, etc.). Note, this is not the actual parameter SQL type in the database.	Req
Parameter Name	The name of the parameter in the stored procedure to which this field is bound.	Req
Parameter Type	The stored procedure parameter type: IN, IN_OUT, OUT, RESULT, RETURN, and UNKNOWN.	Req
Name	The logical name of the model field.	Req
SQL Type	The SQL datatype (from `java.sql.Types`) of the parameter: VARCHAR, TIMESTAMP, SMALLINT, etc.	Req

CHAPTER 40

Object Adapter Model

The Object Adapter Model provides access to any object's, or any of its contained objects', fields, bean properties, and/or methods using path expressions that specify *deep* access to object members.

After the Object Class Name property has been set for the model, and if that class is compiled and loadable, the general keypath binding chooser is available for browsing, or for use when binding to display fields on views. Although views may bind to anonymous path expressions, the developer may also create named model fields that act as aliases to complex path expressions to help isolate changes in the object graph from clients of the model. This can be done manually after adding a model field and setting the model field properties, or automatically by invoking the *Browse/Add Object Field Bindings* action.

This component implements `com.iplanet.jato.model.ExecutingModel`, and may have model operations declared in the IDE. These model operations are mapped to the top-level methods on the adapted object class, and only these methods may be exposed as model operations. Although path expressions may invoke a deep method in the object graph, the current implementation only supports methods with zero parameters or string literal parameters. Model operations may be created manually by adding a new model operation and editing the property sheet for the operation name and parameters. Operations may be automatically added using the contextual menu choice *Complete Missing Operations*.

Property Name	Description	Notes
Default Dataset Name	Because there may be more than one dataset (collection) in the adapted object, this property declares which of these datasets will be considered the default. During field binding, any path expression which cross a contained dataset requires a current dataset name to be set for the model. This property allows the developer to specify a dataset name (path expression denoting a contained dataset) which the model will use in the case of a null current dataset name.	
Is Object Array	When true, indicates that the object type being adapted is an array or collection. Default setting is false.	
Name	The class name of the component.	Req
Object Class Name	The fully qualified class name of the object type being adapted.	Req
Object Factory	Allows the developer to specify a JavaBean implementing the ObjectFactory interface. When the adapted object is not set programmatically and an object factory is specified instead, the model will delegate to the object factory to find the adapted object at runtime. The standard object factories allow objects to be retrieved from the standard J2EE request, session, or application scopes.	

Object Adapter Model Design Actions

Complete Missing Operations

Invoking this action will ensure that there are at least a set of model operations representative of the top-level public methods on the adapted object. Please refer to the detailed JavaDocs which describe the storage mechanism for model operations parameters and return values.

Browse/Add Object Field Bindings

Invoking this action opens the binding chooser dialog and allows the developer to explore properties and operations of the object (properties are only available if the object is a JavaBean, and operations are only available if operations have been defined for top-level object methods). The dialog displays the key path expression for the currently selected node in the object graph. Nodes in the object graph which represent datasets have path expressions highlighted and denoted as dataset names. Selecting *OK* will generate a new model field for the currently selected node in the graph. Selecting *Cancel* will exit the dialog without adding a field. Developers may

also highlight the current path expression and copy it to the clipboard. This capability is useful when setting the *Default Dataset Name* property, or the *Primary Dataset Name* property of a TiledView.

Fields

Property Name	Description	Notes
Key Path	An expression describing how to traverse properties, members, and methods on the adapted object graph to arrive at the field's value. This expression will be used at runtime to resolve a logical field name to a physical field value in the object graph. Please refer to the JavaDocs for detailed explanation of the key path expression syntax.	
Name	The logical name of the model field.	Req

Operations

Property Name	Description	Notes
Name	The model operation name.	Req
Operation Name	The name of the public method on the adapted object class.	Req
Operation Parameter	Describes zero or more parameters for the method of this operation.	

CHAPTER 41

Resource Bundle Model

The Resource Bundle Model enables a developer to use a resource bundle to retrieve localized values. Model field names used in this model are the names of these resources. A common use of the resource bundle model is to localize a page by forming any localized content with static text display fields and binding the display fields to a resource bundle model. The locale used by the model to find resources at runtime may be set programmatically. If not set, the model will use the default system locale.

This model component has a model field chooser which will display the available resources if the resource bundle has been set. In addition, this model supports predefined model fields which encapsulate a resource name. In this way, if resource names change in the bundle, clients of the model will be insulated from the resource name change. Another possible use of the model fields would be to dynamically change a field's resource name at runtime without affecting clients bound to the model field.

Property Name	Description	Notes
Bundle Name	The fully qualified resource name of the bundle (for example, "com/sun/jato/Bundle").	Req
Name	The class name of the component.	Req

Fields

Property Name	Description	Notes
Name	The logical name of the model field.	Req
Resource Name	The key in the resource bundle associated with this model. This property value may be null (not set) if the developer wishes to rely on the model field name property to also represent the name of the resource name. If this property value is set, it must match the name of the resource found in the bundle.	

CHAPTER 42

Web Service Model

The Web Service Model (WS model) allows developers to easily execute Web service operations and retrieve/populate the parameters of those operations via display field bindings. The Web service model is a specialized object adapter model which specifically handles JAX-RPC client stubs. The result is a model which adapts to arbitrary RPC-style Web services.

The Web service model is fully configured by the wizard when it is created: all required properties are set and model operations are declared for the methods on the Web services port. Usually the only remaining configuration task is to set the default dataset name, if needed.

Note that when the model is created, JAX-RPC client stubs are generated in the application's `WEB-INF/classes/stubs` directory, and JAX-RPC support libraries are added to the application's `WEB-INF/lib` directory automatically.

Property Name	Description	Notes
Default Dataset Name	Because there may be more than one dataset (collection) in the adapted object, this property declares which of these datasets will be considered the default. During field binding, any path expression which cross a contained dataset requires a current dataset name to be set for the model. This property allows the developer to specify a dataset name (path expression denoting a contained dataset) which the model will use in the case of a null current dataset name.	
JAX RPC Stub Factory	Whereas the object adapter model allows the specification of an arbitrary object factory bean, this model requires an object factory of type `com.iplanet.jato.model.object.JaxRpcStubFactory`. In this case, the object factory helps the Web service model find the request-scoped JAX-RPC stub. This property is set by the Web service model wizard automatically and generally should not be edited.	
Name	The class name of the component.	Req

Fields

Property Name	Description	Notes
Key Path	An expression describing how to traverse properties, members, and methods on the adapted object graph to arrive at the field's value. This expression will be used at runtime to resolve a logical field name to a physical field value in the object graph. Please refer to the JavaDocs for detailed explanation of the key path expression syntax.	
Name	The logical name of the model field.	Req

Operations

Property Name	Description	Notes
Name	The model operation name.	Req
Operation Name	The name of the public method on the adapted object class.	Req
Operation Parameter	Describes zero or more parameters for the method of this operation.	

CHAPTER 43

Directory Search Model

The Directory Search Model allows us to uses a Model as a backing store for the LDAP Query's Result Set. This allows display fields to be bound to Directory attributes, and is a convenient approach when you have directory store and want to leverage automatic binding of these directory store values to a view. The runtime convenience is that the model provided pagination support and the results are scrollable bidirectionally.

Property Name	Description	Notes
Append Multiple Valued Attributes	This takes values True or False. If you want to append the multiple values of a single attribute, set this to True.	Required
Multiple Value Delimiter	This is used as a delimiter to append multiple values. This is ignored if the above property is set to false.	Optional
Root Context	This is the node from which the search is performed. Its value will be something like "`ou=People,dc=sun,dc=com`"	Required
Search Filter	Specify the filer for your search. This should be in he standard Directory search filter format. Eg: "`(cn=anand*)`"	Required

Also there are properties that can be programatically modified. SearchControl, InitialDirContext are few of the attributes that can be set programatically. Please refer to the Javadocs for more information on how to use these attributes.

Directory Search Model Design Actions

Start by creating Fields on the model. Name each field same as the attribute name in the LDAP. Alternatively you can use the "Attribute Name" instead to hold the attribute name and "Name" as just a logical name.

Fields

Property Name	Description	Notes
Attribute Name	This should match the attribute the name	Optional
Name	The logical name of the model field.	Required

How to Use This Component

1. Drag a DirectorySearchModel into your application.
2. Set the properties in the above table.
3. Add fields to the model. The Model Field name should be the same as the attribute name.
4. Multi-valued attributes can be concatenated to generate a single value
5. For each field Model Field Property “attribute” needs to be modified to match the attribute name in the LDAP server.
6. Create a Pagelet (tiled view) and associate this with the DirectorySearchModel.
7. Drag the required fields into your paglet.
8. Drag your paglet into a View Bean.
9. Create a submit button and a text field in your view bean.
10. When the user enters the search filter in the test field, get the value and set it to the model (in the handle request method of the button).
11. Now the model will update itself with the new result set, which will be displayed back to you.

CHAPTER 44

JDBC ResultSet Adapter Model

The JDBC ResultSet Model will adapt to the ResultSet thats being passed to it. It allows the user to set the Field Names and Types during the design time. The Application Developer can map DisplayFields directly to this Model Fields that has been created. Model will work as a adapter to the underlying ResultSet and will support only retrievel and display of data encapsulated in ResultSet. Also this model will not support Insert & Update actions.

Like the bean adapter model, this model could be used programmatically by calling setResultSet() before value binding or like the bean adapter model it may have config properties defined in the component info which look for the ResultSet automatically in as a scoped object.

Property Name	Description	Notes
ResultSet Attribute Scope	It takes Request, Application and Session as values. If "Any" is specified, the attribute is	
ResultSet Attribute Name	The name of the attribute into which the ResultSet is store.	Required when above is set.

JDBC ResultSet Adapter Model Design Actions

Property Name	Description	Notes
Field Name	Name of the column in the table you are trying to connect.	Optional
Name	The logical name of the model field.	Required

All the remaining fields on this model is not required to be filled in.

CHAPTER 45

Client Session Model

A ClientSession Model uses the JATO client session as its backing data store. ClientSession model fields map directly to client session attributes. Unlike most other models, the fields of a ClientSession Model are merely a pass through vehicle for the values to the client session attributes. In other words, setting the value on a field pushes the value immediately into the client session attribute without the need to execute the model to update the actual backing data store. This model has no internal storage it is a facade to the client session attributes.

Property Name	Description	Notes
Name	The class name of the component.	Required

Fields

Property Name	Description	Notes
Attribute Class	Class type of the HTTP session attribute. The HTTP Session Model implementation of setValue() will use this property to coerce value types.	
Attribute Name	The name of the HTTP session attribute to which the model field maps. This property value may be null (not set) if the developer wishes to rely on the model field name property to also represent the name of the HTTP session attribute. If this property value is set, it must match the exact name intended for the session attribute.	
Name	The logical name of the model field.	Required

CHAPTER 46

Basic Command

The Basic Command component is a controller or request handler component. It is a simple structure for creating reusable and extensible request handling objects.

Property Name	Description	Notes
Name	The class name of the component.	Req

CHAPTER 47

Command Chain

The Command Chain component enables a developer to link together two or more command components to be invoked in sequence.

Property Name	Description	Notes
Chained Command Descriptors	An array of command components to be invoked in sequence.	
Name	The class name of the component.	Req

CHAPTER 48

Application Attribute Factory

The Application Attribute Factory is a factory that acquires an object from application scope.

Property Name	Description	Notes
Attribute Name	The name of the attribute used to retrieve the object from application scope.	
Name	The name of the component instance.	Req

CHAPTER 49

Execute Model and Goto Page Command

The Execute Model Goto Page Command (execute and forward command) automatically executes a model and then displays a page within the current application.

Property Name	Description	Notes
Command Class Name	The name of the command class that will handle the request. By default this is set to the standard implementation `com.iplanet.jato.view.command.ExecuteAndForwardCommand`. Developers should only change the setting of this expert property if they wish to set it to a subclass of the standard command class implementation.	
Executing Model Reference	The reference of the model on which the specified operation will be executed.	Req
Name	The name of the component instance.	Req

Property Name	Description	Notes
Model Operation Name	The name of the model operation that is to be executed.	
Target ViewBean Class Name	The class name of the view bean that the will be displayed after the model is executed.	Req
User Parameters	An array of developer-defined parameters which can be any Java type, including a Java expression. The parameters will be passed into the specified command component's `execute()` method via the `CommandEvent` parameter. The parameters specified in this property will be passed as a single reserved parameter within the standard parameter map. This reserved parameter will be keyed as `com.iplanet.jato.view.command.ViewCommandDescriptorBase.PARAM_USER_PARAMETERS and its type will be java.util.Map`. This expert property is only meaningful if the associated expert property *Command Class Name* has also been set to something other than its default value, and the non default class specified in the *Command Class Name* property has been coded to look for the reserved parameter key `com.iplanet.jato.view.command.ViewCommandDescriptorBase.PARAM_USER_PARAMETERS`.	

CHAPTER 50

Execute Model Command

The Execute Model Command automatically executes a model when invoked.

Property Name	Description	Notes
Command Class Name	The name of the command class that will handle the request. By default this is set to the standard implementation `com.iplanet.jato.view.command.ExecuteModelCommand`. Developers should only change the setting of this expert property if they wish to set it to a subclass of the standard command class implementation.	
Executing Model Reference	The reference of the model on which the specified operation will be executed.	Req
Name	The name of the component instance.	Req
Model Operation Name	The name of the model operation that is to be executed.	
User Parameters	An array of developer-defined parameters which can be any Java type, including a Java expression. The parameters will be passed into the specified command component's `execute()` method via the `CommandEvent` parameter. The parameters specified in this property will be passed as a single reserved parameter within the standard parameter map. This reserved parameter will be keyed as `com.iplanet.jato.view.command.ViewCommandDescriptorBase.PARAM_USER_PARAMETERS` and its type will be `java.util.Map`. This expert property is only meaningful if the associated expert property *Command Class Name* has also been set to something other than its default value, and the non default class specified in the *Command Class Name* property has been coded to look for the reserved parameter key `com.iplanet.jato.view.command.ViewCommandDescriptorBase.PARAM_USER_PARAMETERS`.	

CHAPTER 51

Forward Command

The Forward Command uses the servlet RequestDispatcher to forward to a resource within the current application.

Property Name	Description	Notes
Command Class Name	The name of the command class that will handle the request. By default this is set to the standard implementation `com.iplanet.jato.view.command.ForwardCommand`. Developers should only change the setting of this expert property if they wish to set it to a subclass of the standard command class implementation.	
Name	The name of the component instance.	Req
Path	The path to a resource within the current application. Generally this path denotes a servlet, JSP, HTML, or other file.	Req
User Parameters	An array of developer-defined parameters which can be any Java type, including a Java expression. The parameters will be passed into the specified command component's `execute()` method via the `CommandEvent` parameter. The parameters specified in this property will be passed as a single reserved parameter within the standard parameter map. This reserved parameter will be keyed as `com.iplanet.jato.view.command.ViewCommandDescriptorBase.PARAM_USER_PARAMETERS` and its type will be `java.util.Map`. This expert property is only meaningful if the associated expert property *Command Class Name* has also been set to something other than its default value, and the non default class specified in the *Command Class Name* property has been coded to look for the reserved parameter key `com.iplanet.jato.view.command.ViewCommandDescriptorBase.PARAM_USER_PARAMETERS`.	

CHAPTER 52

Goto ViewBean Command

The Goto ViewBean Command displays a page component when invoked.

Property Name	Description	Notes
Command Class Name	The name of the command class that will handle the request. By default this is set to the standard implementation `com.iplanet.jato.view.command.GotoViewBeanCommand`. Developers should only change the setting of this expert property if they wish to set it to a subclass of the standard command class implementation.	
Name	The name of the component instance.	Req
Target ViewBean Class Name	The class name of the view bean that the will be displayed at the conclusion of the request.	Req
User Parameters	An array of developer-defined parameters which can be any Java type, including a Java expression. The parameters will be passed into the specified command component's `execute()` method via the `CommandEvent` parameter. The parameters specified in this property will be passed as a single reserved parameter within the standard parameter map. This reserved parameter will be keyed as `com.iplanet.jato.view.command.ViewCommandDescriptorBase.PARAM_USER_PARAMETERS` and its type will be `java.util.Map`. This expert property is only meaningful if the associated expert property Command Class Name has also been set to something other than its default value, and the non default class specified in the Command Class Name property has been coded to look for the reserved parameter key `com.iplanet.jato.view.command.ViewCommandDescriptorBase.PARAM_USER_PARAMETERS`.	

CHAPTER 53

Include Command

The Include Command component uses the servlet RequestDispatcher to perform an include of a resource within the current application.

Property Name	Description	Notes
Command Class Name	The name of the command class that will handle the request. By default this is set to the standard implementation `com.iplanet.jato.view.command.IncludeCommand`. Developers should only change the setting of this expert property if they wish to set it to a subclass of the standard command class implementation.	
Name	The name of the component instance.	Req
Path	The path to a resource within the current application. Generally this path denotes a servlet, JSP, HTML, or other file.	Req
User Parameters	An array of developer-defined parameters which can be any Java type, including a Java expression. The parameters will be passed into the specified command component's `execute()` method via the `CommandEvent` parameter. The parameters specified in this property will be passed as a single reserved parameter within the standard parameter map. This reserved parameter will be keyed as `com.iplanet.jato.view.command.ViewCommandDescriptorBase.PARAM_USER_PARAMETERS` and its type will be `java.util.Map`. This expert property is only meaningful if the associated expert property *Command Class Name* has also been set to something other than its default value, and the non default class specified in the *Command Class Name* property has been coded to look for the reserved parameter key `com.iplanet.jato.view.command.ViewCommandDescriptorBase.PARAM_USER_PARAMETERS`.	

CHAPTER 54

Redirect Command

The Redirect Command redirects the current request to any internal or external URL using an HTTP 302 redirect response.

Property Name	Description	Notes
Command Class Name	The name of the command class that will handle the request. By default this is set to the standard implementation `com.iplanet.jato.view.command.RedirectCommand`. Developers should only change the setting of this expert property if they wish to set it to a subclass of the standard command class implementation.	
Name	The name of the component instance.	Req
User Parameters	An array of developer-defined parameters which can be any Java type, including a Java expression. The parameters will be passed into the specified command component's `execute()` method via the `CommandEvent` parameter. The parameters specified in this property will be passed as a single reserved parameter within the standard parameter map. This reserved parameter will be keyed as `com.iplanet.jato.view.command.ViewCommandDescriptorBase.PARAM_USER_PARAMETERS` and its type will be `java.util.Map`. This expert property is only meaningful if the associated expert property *Command Class Name* has also been set to something other than its default value, and the non default class specified in the *Command Class Name* property has been coded to look for the reserved parameter key `com.iplanet.jato.view.command.ViewCommandDescriptorBase.PARAM_USER_PARAMETERS`.	
URL	The URL to which to redirect the request.	Req

CHAPTER 55

Regular Expression Validator

The Regular Expression Validator is a simple validator that uses JDK 1.4 regular expressions to validate a value.

Property Name	Description	Notes
Name	The name of the component instance.	Req
Validation Rule	The JDK 1.4 regular expression used for validation. Before being validated, data is converted to a string using the `com.iplanet.jato.util.TypeConverter` class.	Req

CHAPTER 56

Request Attribute Factory

The Request Attribute Factory is a factory that acquires an object from request scope.

Property Name	Description	Notes
Attribute Name	The name of the attribute used to retrieve the object from request scope.	
Name	The name of the component instance.	Req

CHAPTER 57

Session Attribute Factory

The Session Attribute Factory is a factory that acquires an object from session scope.

Property Name	Description	Notes
Attribute Name	The name of the attribute used to retrieve the object from session scope.	
Name	The name of the component instance.	Req

CHAPTER 58

Simple Choice

A simple Choice implementation.

Property Name	Description	Notes
Label	The label of the choice that is displayed to the end user.	Req
Name	The name of the component instance.	Req
Value	The value of the choice.	Req

CHAPTER 59

Model Reference

A Model Reference configures an instance of `com.iplanet.jato.model.SimpleModelReference`.

Property Name	Description	Notes
Instance Name	The name of the model instance within this request. If no instance name is specified, the default instance will be used. All references that specify the same instance name (including the default) will share the same model instance.	
Look in Session	Determines whether the model will be obtained from the HTTP session. If this value is `true`, but the model is not available from the session, a new model instance will be created and stored in the session if the *Store In Session* property is set to `true`.	
Model Class Name	The fully qualified name of the model class.	Req
Name	The name of the component instance.	Req
Store in Session	Determines whether the model will be stored in the HTTP session if a new instance of the model is created (a new instance my not be created if the *Look in Session* property is set to `true`). The model will be stored in the session using the specified instance name, or the default instance name.	

CHAPTER 60

Type Validator

A Type Validator validates based on successful conversion to a specified type.

Property Name	Description	Notes
Name	The name of the component instance.	Req
Validation Rule	The fully qualified class name used for validation. During validation, the provided value is converted to this type using the `com.iplanet.jato.util.TypeConverter` class. Validation fails if conversion to this type fails.	Req

CHAPTER 61

User-Defined Command

The User-defined Command component represents a reference to any command component within the current application or its component libraries.

Property Name	Description	Notes
Command Class Name	The fully qualified name of the command class.	Req
Name	The name of the component instance.	Req
Operation Name	The name of the operation that will be passed into the command's `execute()` method via the `CommandEvent` parameter.	
Parameters	An array of developer-defined parameters which can be any Java type, including a Java expression. The parameters will be passed into the specified command component's `execute()` method via the `CommandEvent` parameter.	

CHAPTER 62

WebAction Command

The Web Action Command invokes a WebAction on the specified WebActionHandler component.

Property Name	Description	Notes
Command Class Name	The name of the command class that will handle the request. By default this is set to the standard implementation `com.iplanet.jato.view.command.WebActionCommand`. Developers should only change the setting of this expert property if they wish to set it to a subclass of the standard command class implementation.	
Name	The name of the component instance.	Req
Operation Name	The WebAction to perform.	Req
WebAction Handler Path	A qualified path that indicates which WebActionHandler visual component to invoke. This path is resolved relative to the parent of the command field component that invoked this command. For example, if the developer associates a button in a container view component with this command, the WebAction would be invoked on the parent container view. The syntax of this path follows the standard view name path expression syntax, using forward slashes ("/") as delimiters. All components in the path except the last must refer to a ContainerView or a derivative of ContainerView (such as TiledView). Both relative and absolute paths are possible. If a name path begins with a forward slash, the name is assumed to be relative to the page (the ViewBean). If the path does not begin with a forward slash, the name is assumed to refer to a child relative to the current container. Two dots ("..") may be used to refer to the container that is the parent of the current container.	

Index