Sun™ ONE Application Framework Component Reference Guide

Sun™ ONE Studio 5 update 1

Sun Microsystems, Inc.
www.sun.com

Part No. 817-4661-10
October 2003, Revision A

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

Contents

Preface

This Sun™ ONE Application Framework Component Reference Guide introduces the components in the Sun ONE Application Framework Library, which fall into four basic groups: Visual components, Model Components, Command Components, and Non-Visual Components.

How This Book Is Organized

Chapter 1 Overview introduces the four basic component groups: Visual Components, Model Components, Command Components, and Non-Visual Components.

Chapter 2 describes the Basic Container View (Pagelet) component and its properties.

Chapter 3 describes the Basic Tiled View component and its properties.

Chapter 4 describes the Basic Tree View component and its properties.

Chapter 5 describes the Basic ViewBean (Page) component and its properties.

Chapter 6 describes the Button component and its properties.

Chapter 7 describes the Check Box component and its properties.

Chapter 8 describes the Combo Box component and its properties.

Chapter 9 describes the Data-Driven Combo Box component and its properties.

Chapter 10 describes the Data-Driven List Box component and its properties.

Chapter 11 describes the Data-Driven Radio Buttons component and its properties.

Chapter 12 describes the File Upload component and its properties.

Chapter 13 describes the Hidden Field component and its properties.

Chapter 14 describes the Hyperlink (HREF) component and its properties.

Chapter 15 describes the Image component and its properties.

Chapter 16 describes the List Box component and its properties.

Chapter 17 describes the Password Field component and its properties.

Chapter 18 describes the Radio Buttons component and its properties.

Chapter 19 describes the Static Text Field component and its properties.

Chapter 20 describes the Text Field component and its properties.

Chapter 21 describes the Text Area component and its properties.

Chapter 22 describes the Validating Text Field component and its properties.

Chapter 23 describes the Validating Text Area component and its properties.

Chapter 24 describes the Bean Adapter Model component and its properties.

Chapter 25 describes the Custom Model component and its properties.

Chapter 26 describes the Simple Custom Model component and its properties.

Chapter 27 describes the Custom Tree Model component and its properties.

Chapter 28 describes the HTTP Session Model component and its properties.

Chapter 29 describes the JDBC SQL Query Model component and its properties.

Chapter 30 describes the JDBC Stored Procedure Model component and its properties.

Chapter 31 describes the Object Adapter Model component and its properties.

Chapter 32 describes the Resource Bundle Model component and its properties.

Chapter 33 describes the Web Service Model component and its properties.

Chapter 34 describes the Basic Command component and its properties.

Chapter 35 describes the Command Chain component and its properties.

Chapter 36 describes the Application Attribute Factory component and its properties.

Chapter 37 describes the Execute Model and Goto Page Command component and its properties.

Chapter 38 describes the Execute Model Command component and its properties.

Chapter 39 describes the Forward Command component and its properties.

Chapter 40 describes the Goto ViewBean Command component and its properties.

Chapter 41 describes the Include Command component and its properties.

Chapter 42 describes the Redirect Command component and its properties.

Chapter 43 describes the Regular Expression Validator component and its properties.

Chapter 44 describes the Request Attribute Factory component and its properties.

Chapter 45 describes the Session Attribute Factory component and its properties.

Chapter 46 describes the Simple Choice component and its properties.

Chapter 47 describes the Model Reference component and its properties.

Chapter 48 describes the Type Validator component and its properties.

Chapter 49 describes the User-Defined Command component and its properties.

Chapter 50 describes the WebAction Command component and its properties.

Using UNIX Commands

This document might not contain information on basic UNIX® commands and procedures such as shutting down the system, booting the system, and configuring devices. See the following for this information:

- Software documentation that you received with your system
- Solaris™ operating environment documentation, which is at

  `http://docs.sun.com`

Related Documentation

Application	Title	Part Number
Sun ONE Application Framework 2.1	*Sun ONE Application Framework Overview, Sun ONE Studio 5 update 1*	817-4360-10
Sun ONE Application Framework 2.1	*Sun ONE Application Framework Tutorial, Sun™ ONE Studio 5 update 1*	817-4358-10
Sun ONE Application Framework 2.1	*Sun ONE Application Framework IDE Guide, Sun ONE Studio 5 update 1*	817-4104-10
Sun ONE Application Framework 2.1	*Sun ONE Application Framework Developer's Guide, Sun ONE Studio 5 update 1*	817-4359-10
Sun ONE Application Framework 2.1	*Sun ONE Application Framework Component Author's Guide, Sun ONE Studio 5 update 1*	817-4362-10
Sun ONE Application Framework 2.1	*Sun ONE Application Framework Tag Library Reference, Sun ONE Studio 5 update 1*	817-4361-10

Accessing Sun Documentation

You can view, print, or purchase a broad selection of Sun documentation, including localized versions, at:

`http://www.sun.com/documentation`

Contacting Sun Technical Support

If you have technical questions about this product that are not answered in this document, go to:

`http://www.sun.com/service/contacting`

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. You can submit your comments by going to:

`http://www.sun.com/hwdocs/feedback`

Include the title and part number of your document with your feedback:

Sun ONE Application Framework Component Reference Guide, part number 817-4661-10

CHAPTER 1

Component Overview

The components in the Sun ONE 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 Sun ONE 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 Sun ONE 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	A specialization 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.
File Upload	A component that provides a way for users to send files to the server.
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.
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.

Component Name	Description
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.
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.
Custom Model	An arbitrary implementation of the `com.iplanet.model.Model` interface.
Simple Custom Model	A model that provides a foundation for an arbitrary new model which requires advanced dataset management and pagination support.
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.
HTTP Session Model	A model that uses the HTTP session as its backing datastore.
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.
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`.

Component Name	Description
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

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 25

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 **26**

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 27

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 **28**

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 Sun ONE 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 29

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 30

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 31

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 **32**

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 **33**

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 34

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 35

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 36

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 **37**

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 **38**

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 39

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 **40**

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 41

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 42

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 43

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 44

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 45

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 46

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 **47**

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 48

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 49

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 50

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