

Websydian™ WebClient

Template commands

v1.4r4

This documentation and related computer software program (hereinafter referred to as "Documentation") is for the end user's informational purposes only and is subject to change or withdrawal by ADC Austin, Inc. ("ADC") at any time. Documentation may not be copied, transferred, reproduced, disclosed, or duplicated, in whole or in part, without the prior written consent of ADC. Documentation is proprietary information of ADC and protected by the copyright laws of the United States and international treaties. Notwithstanding the foregoing, licensed users may print a reasonable number of copies of Documentation for their own internal use, provided that all ADC copyright notices and legends are affixed to each reproduced copy. Only authorized employees, consultants, or agents of the user who are bound by the confidentiality provisions of the license for the software are permitted to have access to such copies. To the extent permitted by applicable law, ADC provides Documentation "as is" without warranty of any kind, including without limitation, any implied warranties of merchantability, fitness for a particular purpose or noninfringement. In no event will ADC be liable to the end user or any third party for any loss or damage, direct or indirect, from the use of Documentation, including without limitation, lost profits, business interruption, goodwill, or lost data, even if ADC is advised of the possibility of such loss or damage. The use of any product referenced in Documentation and Documentation is governed by the end user's applicable license agreements. The manufacturer of Documentation is ADC Austin, Inc. © 2008 ADC Austin, Inc. All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

Table of Contents

| | |
|------------------------------------|----|
| Chapter 1: Customizing WebClient | 5 |
| Attach point | 5 |
| Bean controls | 5 |
| Control action | 5 |
| Control context | 6 |
| Control name property | 7 |
| Control reference | 7 |
| Control template | 7 |
| Default attach points for controls | 8 |
| Name identifier | 8 |
| Output path | 9 |
| Page template | 9 |
| Physical event | 9 |
| Property modifier | 10 |
| Root template | 10 |
| Source | 11 |
| Template generation | 11 |
| Appendix A: Template Commands | 13 |
| -- comment | 13 |
| ALERTS | 13 |
| Action | 13 |
| ActionArg | 13 |
| AttachPoint | 13 |
| ButtonGroup | 14 |
| ButtonValue | 14 |
| CheckedValue | 14 |
| DataType | 14 |
| DecLength | 15 |
| Edit | 15 |
| EditNameID | 15 |
| EditProperty | 15 |
| Embed | 15 |
| EnableAction | 16 |
| Event | 17 |
| FOCUS | 18 |
| FindControl | 18 |

| | |
|--|----|
| GridEnd | 18 |
| If | 18 |
| IgnoreActions | 19 |
| ImplName | 19 |
| Length | 19 |
| NameID | 19 |
| OptSelected | 20 |
| OptText | 20 |
| OutTo | 20 |
| Owner | 20 |
| Owner | 20 |
| Panel | 21 |
| Param | 21 |
| Parent | 21 |
| Property | 21 |
| Render | 21 |
| Site | 21 |
| State | 22 |
| This | 22 |
| This | 22 |
| This | 22 |
| Title | 22 |
| Type | 22 |
| UncheckedValue | 23 |
| Visible | 23 |
| WsRes | 23 |
| getVar(String var) | 23 |
| jsonColVis | 23 |
| jsonCols | 24 |
| jsonImplNames | 24 |
| jsonOptions | 24 |
| jsonRows | 24 |
| jsonSelected | 24 |
| jsonValues | 24 |
| processAction(String actionName [, Object arg]) | 25 |
| setVar(String var, Object val) | 25 |
| value | 25 |

Chapter 1: Customizing WebClient

Attach point

A template for a given screen can be generated from more than one template by using attach points. The templates are arranged in order from root to leaf. An attach point is a location defined in a template that other templates further towards the leaf can populate with data. See [Template generation](#).

Attach points can be defined in the [root template](#) or a [page template](#) using the `/(!AttachPoint)` command. When an attach point is defined, it is initially empty (it will not expand to anything), until it is populated by templates further down the inheritance path, or by controls via their [control name property](#).

See Also

`/(!AttachPoint)`

Bean controls

Some controls are implemented in Plex as a Java Bean control. Although Java Beans cannot be loaded in WebClient, their functionality can be emulated through the use of a custom template and customized source code objects in Plex.

The name property of the bean control is the default name of template that is used (without the `.ctrl` extension.)

In the source code objects, variables in the control may be created using names which you define. The methods that do this are called `setVar` and `getVar` and they are located in the class `WebBeanData`. The source code objects may also call the method `processAction` to send a [control action](#) to the template.

In the bean control's template, the variables may be retrieved using the `/(!Var)` control command.

If the bean control sends a value to the server as a request parameter, the data in the parameter is taken to be a JSON object, and bean's variables are updated with the keys and values of that object.

Example

See `DojoTab.ctrl` in the distribution for an example of a tab control implemented in Javascript.

Control action

If the value of a control changes, the rendering on the web browser needs to be updated. WebClient supports this two different ways. In the first method, page refresh, when the value of a control changes, the web browser is instructed to reload the page from the server. This will cause the template to be re-evaluated and the user will see the new data. This is the default method that is used to update the display.

In order to avoid page reloads, a second method is supported. In this method, called action updates, the page is not reloaded. Instead, the server sends back commands corresponding to only the changes that have occurred. These are called actions. An

action has a name and an optional argument. Each action is expanded into a block of Javascript code that is then used to incrementally update the screen.

In order to use action updates with a control, the desired actions must be enabled for that control. This is accomplished by using the `/(!EnableAction)` template command within the [control template](#) for that control. The names of the actions that are going to be handled by the template are specified as parameters to that command. Then the `/(!Action)` command can be used to specify that commands that should be expanded when that action occurs. Ordinarily, the `/(!Action)` command is placed within an [attach point](#) that is within a Javascript [output path](#).

The following actions are defined:

`SetValue`

sent when the value of a field has been changed.

`SetState`

sent when the state of the control has been changed.

`SetFocus`

sent when the control is to be given focus.

`Refresh`

sent to a panel or site control when the panel needs to be refreshed. The argument is the panel control.

`Alert`

sent when a Dialog Message or Status Message has been sent to the panel.

`Enquiry`

sent when an Enquiry Message has been sent to the panel.

`Load`

sent to a site control when the child panel has been loaded. The argument is the panel control.

`Hide`

sent to a site control when the child panel has been hidden.

`Show`

sent to a site control when the child panel has been shown.

`Grid.Insert`

sent to a grid when a row has been inserted. The argument is the row.

`Grid.Clear`

sent to a grid when the grid has been cleared.

`Grid.Remove`

sent to a grid when a row has been removed. The argument is the row.

`Grid.Update`

sent to a grid when a row has been updated. The argument is the row.

Control context

When the template generator is in a control context, it is expanding a template for a specific control. This means it is possible to use control commands to access that control.

The template generator is automatically in a control context when it is expanding a [control template](#). It is also in a control context when expanding the `/(FOCUS)` command.

Control name property

The control name property is a panel property that can be set in Plex for each control. Although it is not required to be set, it is strongly recommended that the control name be set for each control that is to be used under WebClient.

The control name property can take two syntaxes. The first syntax is simply a unique name given to the control. This name appears in the generated HTML and Javascript and aids in debugging. In the second syntax, the unique name is followed by a colon, then an [attach point](#) name, then optionally another colon and template parameters. In this case, the control is given a name and also a location in the template where it is to appear. The control will be placed in the named attach point, which must be defined in the current panel's [root template](#) or [page template](#). Any additional parameters following the attach point are passed to the control's template.

Example

If you have defined an attach point named `ButtonsArea`, and you want to place a button in it, you might assign the button the control name `ExitButton:ButtonsArea`.

Control reference

Certain template commands are not intended to expand to text, but rather are used to refer to a control to which another command is addressed. The next parameter of the command is taken to be the next command to be applied.

Any control with a control name can be used as a control reference, by giving the control name as a command. In addition, there are special template commands which are useful as control references.

Example

If you have a control named `Grid1` and wish to access its size, you can may write `/(!Grid1:Size)`. Here, `Grid1` is functioning as a control reference command. If you have a column heading control and need the `NameID` of the corresponding edit control, you can write `/(!Edit:!NameID)`. `/(!Edit)` is a control reference command.

Control template

A control template is used to produce HTML and Javascript for a control that is on a Plex panel. A control template is expanded and placed in the [attach point](#) that is associated with the control via the [control name property](#).

The default control template that is used is in a file that is named by taking the type for of the control and prefixing it by `web`. The extension that is used is `.ctrl`. For example, the default control template used for edit fields is `WebEdit.ctrl`. These control files are normally located in the `SysTemplates` folder.

To override and specify a specific control template that is to be used for a control, specify the `template=` parameter in the control name property for that control. The value of this parameter will be suffixed with the extension `.ctrl` and used instead of the default control template.

Example

If you have a custom template called `ToggleButton.ctrl` and wish to use this template for a check box control, you may set the control name parameter `template=ToggleButton`.

Default attach points for controls

If the [control name property](#) for a control does not specify an [attach point](#) for the control, a default attach point is selected based on the type of the control. The attach point is selected as follows:

Column heading

`ColumnsArea`

Grid edit control

`CellsArea`

Menu bar

`MenuBarArea`

Menu

`MenuArea`

Menu item or separator

`MenuItemArea`

Toolbar

`ToolBarArea`

Toolbar Button

`ToolArea`

All other controls

`MainArea`

In addition, the parameter `default` is passed to the template when the default attach point is selected.

Example

If an edit field control is given a control name property of `MyControl`, the default attach point of `MainArea` applies and the parameter `default` is also passed to the template. In other words, it is as if the edit field control had a control name property of `MyControl:MainArea:default`.

Name identifier

If a Plex panel contains child panels, the control name of a control, as given by `/(!Name)`, may not uniquely identify a control in the web browser. This is because even though the control is unique within a page, it still may appear in multiple child panels and create duplicates.

To resolve this issue, use the `/(!NameID)` command to generate a unique name identifier. This name will be prefixed by both the site's control name and the child panel's name. This name is suitable for use as an identifier in the web browser's DOM.

Output path

WebClient can send a response back to the client of several different types. The output path is a parameter that determines what type of response is sent. In addition, some response formats can include hierarchical data. The output path can be used to index into that data.

The main output paths supported are:

`html`

the response mime-type is `text/html`. The contents are escaped according to HTML rules.

`json`

the response mime-type is `application/json`. The contents are encoded as a valid JSON expression.

For the `json` output type, a hierarchical naming scheme is possible. After the `json` component, which must be the top-level component in the path, a dot-separated path may be placed. The components of this path are created as JSON sub-objects in the JSON path.

Example

Suppose a response is created with the command `/(!OutTo:json.x.y.z)abc/(!OutTo)`. If this response is returned to a web browser and loaded as a JSON expression and assigned to the variable `t`, then `t.x.y.z == "abc"`.

Page template

A page template is an optional template that is expanded during the template generation process. A page template can place data into [attach points](#) that are defined by either the [root template](#) or another page template.

See Also: [Template generation](#)

Physical event

The Plex physical events which may be handled in WebClient are:

- Pressed
- Load Grid
- Double Click
- Select
- Mouse Down
- Mouse Move
- Mouse Up

- Gained Focus
- Lost Focus
- Changed
- Updated
- Select Menu
- Query Close
- Modified

See Also

`/(!Event)`

Property modifier

A property modifier is an optional parameter which may be placed after the property name parameter in a `/(!Property)` command. The following property modifiers are available:

`x`

for a property value which is a coordinate in the format `x y`, expands to `x`.

`y`

for a property value which is a coordinate in the format `x y`, expands to `y`.

`hexcolor`

for a property value which is a color in the format `r g b`, expands to the color in HTML `#rrggbb` notation.

`htmlaccel`

for a property value which uses the `&` character to indicate that the next character is to be underlined, converts this notation to HTML using the `<u>` tag.

`htmlbr`

for a property value which uses the `^` character to indicate a line break, converts this notation to HTML using the `
` tag.

`nbspifempty`

if the property value is blank or consists only of spaces, expands to ` `; instead (as HTML [source](#)). Otherwise, leaves the value as is.

`html`

marks that the value is to be treated as HTML [source](#).

`js`

marks that the value is to be treated as JavaScript [source](#). This is the default option; if `html` is not specified, the property command will generate JavaScript source.

Root template

The root template is first template that is expanded during the template generation process. The default root template is called `webShell-root.wcli`. It expands to the basic HTML and Javascript templates where all other template data is placed, via [page templates](#) and [control templates](#).

See Also: [Template generation](#)

Source

Text that is expanded into the template can be of two types, source or data. By default, any text that is placed into a template at run-time is considered data. When text is considered data, it cannot contain markup commands such as HTML tags or Javascript expressions. If the data contains any special characters that would have meaning to the HTML or Javascript parser, such as angle brackets, ampersands, or quote markers, they will be automatically escaped appropriately for those parsers.

Specifically, if text is handled as data, and the [output path](#) is set to an HTML fragment, the characters less-than, greater-than, and ampersand are escaped with entity references. For example, `<foo>` is encoded as `<foo>`

If text is handled as data, and the output path is a Javascript fragment, the text is encoded as a valid Javascript string literal constant. This includes all quotation marks automatically. For example, `abc"def` is encoded as `"abc\"def"`.

If you have some text that is to be expanded at run-time and interpreted as an actual HTML tag or Javascript expression, with escaping, you must inform WebClient to handle it as source. This can be accomplished with a [property modifier](#) parameter on the property command.

Template generation

When generating a template for a panel, WebClient first attempts to locate a [root template](#). The root template is a template available in the classpath such that the following two conditions are met:

- The name of the file consists of the implementation name of a function followed by `-root.wcli`.
- That function is in the inheritance path of the panel function being generated.

When there are two or more such functions in the inheritance path, the lowest (closest to the generated function) one takes precedence. If no root templates are available, template generation aborts with an error.

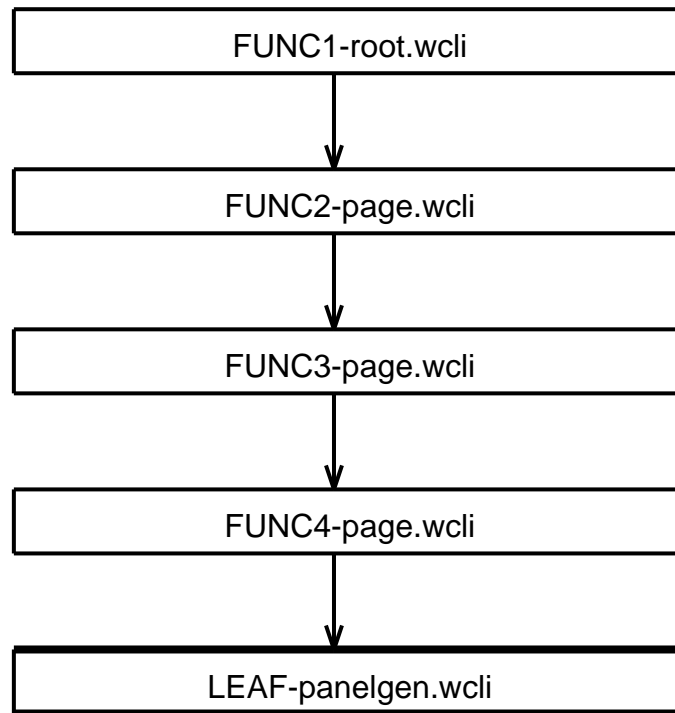
WebClient loads the root template, then it searches for [page templates](#) which may occur in the inheritance path between the generated function and the root template function. The following must be true for a valid page template:

- The name of the template file consists of the implementation name of a function followed by `-page.wcli`.
- That function is in the inheritance path of the panel function being generated, and is lower than the root template that is in use.

The page templates are applied in order starting from the page template closest to the root template, and ending with the template closest to the panel function. Page templates are optional.

Finally, WebClient automatically generates a template for the panel function itself. This file does not need to be supplied by the user and is named by taking the implementation name of the panel function and appending `-panelgen.wcli`. This file is obtained by automatically expanding all of the [control templates](#) in the panel.

The following figure illustrates the process:



Appendix A: Template Commands

-- comment

Synopsis

```
/(!-- comment)
```

Description

This command is a server-side comment. It has no effect.

ALERTS

Synopsis

```
/(ALERTS)
...
/(ALERTS)
```

Description

The contents of the data inside the `/(ALERTS)` block is expanded once for every outstanding alert message associated with the panel. The details of each message can be retrieved within the block by using the `/(AlertMsg)`, `/(AlertType)`, and `/(AlertFdbk)` commands.

Action

Synopsis

```
/(!Action:[control action])
...
/(!Action)
```

Description

Allows a template to handle [control actions](#). The action must have been previously enabled with the `/(!EnableAction)` command. The data inside of the `/(!Action)` command is expanded for every action of that type which has been sent to the command.

For an example, see the entry for `/(!EnableAction)`.

ActionArg

Synopsis

```
/(!ActionArg)
```

Description

The command is valid within the body of an `/(!Action)` command. When specified, it expands to the argument of the [control action](#) being processed, if any. The value of the argument depends on the type of action. For more information, see the documentation specific for the control action.

AttachPoint

Synopsis

```
/(!AttachPoint:name=[[attach point]]
  [ :inlinetemplate ]
  [ :sep=separator]
  [ :args...])
[ ... inline template data ...
  /(!AttachPoint) ]
```

Description

Defines an [attach point](#) in the current template where future templates may populate. The attach point is identified by the given name. The `/(!AttachPoint)` command expands to an empty string; however, the position is marked and when future templates invoke the attach point, their output will be placed at that position. The attach point is invoked by using a command with the same name as the attach point.

If the `inlinetemplate` flag is set, the `AttachPoint` command accepts a block, and this block is expanded as a template whenever the attach point is used. If the `inlinetemplate` flag is not present, the `AttachPoint` command must not have a block.

If the `sep` parameter is specified, it specifies text that is used as a separator between invocations of the attach point. The separator text is placed in between every invocation.

Any other parameters that are specified become default arguments for the templates attached to the attach point.

ButtonGroup

Synopsis

```
/(!ButtonGroup)
```

Description

Expands to the control name of the current button group. The button group is the control name that is associated with the current group of radio buttons.

ButtonValue

Synopsis

```
/(!ButtonValue)
```

Description

Expands to the button value of radio buttons.

CheckedValue

Synopsis

```
/(!CheckedValue)
```

Description

Expands checked value of the check box field.

DataType

Synopsis

```
/(!DataType)
```

Description

Expands to the data type of the field which is bound to the control. The value will be one of the following:

- Char
- FixedDec
- Long
- Date
- Time
- Object

DecLength

Synopsis

```
/(!DecLength)
```

Description

Expands to the design decimal places of the field which is bound to the control.

Edit

Synopsis

```
/(!Edit)
```

Description

Restrict: WebColumnHeadingData

References the column's associated edit control. This is a [control reference](#) command.

EditNameID

Synopsis

```
/(!EditNameID)
```

Description

Expands to the fully qualified [name identifier](#) of the edit control that is associated with the current control, which must be a Column Heading control.

EditProperty

Synopsis

```
/(!EditProperty:[[property name]] [ :[[property modifier]] ] )
```

Description

Works similarly to `/(!Property)`, except the properties are retrieved from the edit control that is associated with the current control, which must be a Column Heading control.

Embed

Synopsis

```
/(Embed:output type [:cdata] )
...
/(Embed)
```

Description

The contents of the Embed block are expanded and encoded with the specified output type. This is used when one output type is embedded in another; for example, JavaScript within HTML, or HTML within JavaScript.

The output type can be:

```
html
```

Encodes the output as HTML.

```
js
```

Encodes the output as a Javascript expression.

If the flag `cdata` is specified, and the current output type is HTML, then the current HTML element is assumed to be of type CDATA, and entity encoding is bypassed as is necessary for CDATA elements.

Example

The following illustrates retrieving the value of `MyField` into a Javascript variable, using a script embedded within an HTML page:

```
<script type="text/javascript">
/(Embed:js:cdata)
  var fld_value = /(MyField);
...
/(Embed)
</script>
```

The following illustrates generating a snippet of HTML from within a Javascript function:

```
function my_function() {
  var html = "<span class='myclass'>"
            + /(Embed:html)/(MyField)/(Embed)
            + "</span>";
  ...
}
```

EnableAction

Synopsis

```
/(!EnableAction:[[control action]] [ ,[[control action]]... ])
```

Description

When specified for a control, indicates that the template will handle the given [control actions](#). Any number of actions may be specified, separated by commas. Once an action is enabled, the runtime will not automatically refresh the page when an action of that type is processed. Instead, it will enqueue the action to a list associated with the control. The template must then expand the action list by using the `/(!Action)` command.

Example

The following example shows how a read-only field can change its value without refreshing the screen.

```
/(!-- The field is mapped as an HTML SPAN element.)
<span id="/(!NameID)">
  /(!This)
</span>

/(!-- Indicate that when the SetValue control
      action is processed, we will handle it here.)
/(!EnableAction:SetValue)

/(!-- Place Javascript code inside of the JS
      attach point to perform the update.)
/(!JS)

/(!-- The code in the following block will be run
      as Javascript when the value of the control
      is changed.)
/(!Action:SetValue)
  document.getElementById("/(!NameID)").innerText
    = /(!This);
/(!Action)

/(!JS)
```

Event

Synopsis

```
/(!Event:phys=[[physical event]])
```

Description

If the given physical event is mapped on this control, expands to its mapped event number. Otherwise, expands to the empty string.

Example

The following template will expand differently depending on whether or not the `Pressed` event is mapped to the current control.

```
/(!If:/(!Event:phys=Pressed))
    The pressed event is mapped
/(!Else)
    The pressed event is not mapped
/(!If)
```

FOCUS

Synopsis

```
/(FOCUS)
...
/(FOCUS)
```

Description

The contents of the data inside the `/(FOCUS)` block is expanded once for each control on the panel that has received initial focus. The block is expanded with a valid [control context](#) for each focused control.

FindControl

Synopsis

```
/(!FindControl: [[control id]])
```

Description

This [control reference](#) command yields the control of the panel that has the specified Plex control ID. The control ID is a numeric value that is assigned by the Plex panel generator. It is unstable (changes with every generation) and is included in certain control properties, which is why you may need this command.

GridEnd

Synopsis

```
/(!GridEnd)
```

Description

Expands to `Y` if the Plex program has executed `Set Grid End On` for this grid. Otherwise, expands to nothing.

If**Synopsis**

```
/(!If:value)
...
[ /(!Else) ]
...
/(!If)
/(!If:value1=value2)
...
[ /(!Else) ]
...
/(!If)
```

Description

This command performs a comparison. It has two forms. In the first form, the contents of the `/(!If)` block are expanded if `value` expands to a non-empty string. In the second form, the contents are expanded if `value1` is equal to `value2`. In both forms, an optional `/(!Else)` clause may be specified. The `/(!Else)` block is terminated by the `/(!If)` command terminator; it does not have its own terminator.

IgnoreActions**Synopsis**

```
/(!IgnoreActions)
```

Description

When specified for a control, causes the control to ignore any [control actions](#) that are sent to it. This command always expands to the empty string.

This is useful for a template which does not respond to actions, to prevent the screen from refreshing.

ImplName**Synopsis**

```
/(!ImplName)
```

Description

Expands to the implementation name of the field which is bound to the control.

Length**Synopsis**

```
/(!Length)
```

Description

Expands to the design length of the field which is bound to the control.

NameID

Synopsis

```
/(!NameID)
```

Description

Expands to the fully qualified [name identifier](#) of the current control. The NameID is unique even when child panels are used. (The control name itself in such a situation need not be unique.)

OptSelected

Synopsis

```
/(!OptSelected:[option])
```

Description

Expands to Y if the option selected is the one indicated.

OptText

Synopsis

```
/(!OptText)
```

Description

Returns the text of the current option of a combobox. This command is valid within the `/(!This)` command block of a combobox control.

OutTo

Synopsis

```
/(OutTo:[output path])
...
/(OutTo)
```

Description

The contents of the OutTo block are placed into the response stream at the specified [output path](#). The OutTo command itself expands to nothing; all data is placed in the path.

Owner

Synopsis

```
/(!Owner)
```

Description

Returns the owning panel.

Owner

Synopsis

```
/(!Owner)
```

Description References the panel that owns the current control. This is a [control reference](#) command.

Panel

Synopsis `/(!Panel)`

Description This [control reference](#) command refers to the panel control. This is the root element of the control hierarchy.

Param

Synopsis `/(!Param:[parameter name])`

Description Expands to the value of the given parameter in the current template. If the current template is a [control template](#), the parameter may be a control name parameter. If the current template is an inline template, the parameter may be a parameter specified in the inline template invocation.

Parent

Synopsis `/(!Parent)`

Description References the parent of the control in the control hierarchy. This is a [control reference](#) command.

Property

Synopsis `/(!Property:[property name] [:[property modifier]])`

Description Expands to the value of the specified property of the current control. If the property is not defined, expands to the empty string. An optional [property modifier](#) may be specified.

Render

Synopsis `/(!Render:[output path] [:args...])`

Description Performs rendering of the current panel, and sends the result of rendering the panel to the specified [output path](#). If any optional arguments are specified, they are carried over as request parameters and may be retrieved with the `/(!Req)` command.

Site

- Synopsis** `/(!Site)`
- Description** Expands to the panel object that is loaded in the current site control.

State

- Synopsis** `/(!State)`
- Description** Expands to the current control's state value, as an integer.

This

- Synopsis** `/(!This) ... /(!This)`
- Description** Expands to the contents of the block, repeated once for every option in the combo box control.

This

- Synopsis** `/(!This) ... /(!This)`
- Description** Expands the contents of the command block once for every grid row in the grid.

This

- Synopsis** `/(!This)`
- Description** Expands to the value of the current control. This command also works as a [control reference](#), referring to the current control.
- The behavior of this command depends on which type of control is being used. See the control-specific documentation for details.

Title

- Synopsis** `/(!Title)`
- Description** Expands to the title of the current panel.

Type

Synopsis

```
/(!Type)
```

Description

Returns the type of the current control. This is the value from the Type property in the panelresource file.

UncheckedValue

Synopsis

```
/(!UncheckedValue)
```

Description

Expands unchecked value of the check box field.

Visible

Synopsis

```
/(!Visible)
```

Description

Expands to Y if the current panel is visible. Otherwise, expands to nothing.

WsRes

Synopsis

```
/(!WsRes)
```

Description

At runtime, expands to the web resources path that associated with the web library id of the current template. The web library id is defined by the project containing the template; that id is then mapped to a URL resource path by a runtime-specific configuration mechanism.

Example

The following HTML image element refers to an image which is located in the images directory under the current project's web resources path:

```
</img>
```

getVar(String var)

Synopsis

```
getVar(String var)
```

Description

Retrieves the value of the specified bean variable in the current control.

jsonColVis

Synopsis

```
/(!jsonColVis)
```

Description

Returns a list of the visibilities of each column on the grid, as a Javascript array. Each element is either true if the column is visible, or false if it is not.

jsonCols

Synopsis

```
/(!jsonCols)
```

Description

Returns a list of the names of the columns of the grid, as a Javascript array.

jsonImplNames

Synopsis

```
/(!jsonImplNames)
```

Description

Returns a list of the implementation names of the column fields of the grid, as a Javascript Array.

jsonOptions

Synopsis

```
/(!jsonOptions)
```

Description

Returns a list of the names of the options in the combo box, as a Javascript list.

jsonRows

Synopsis

```
/(!jsonRows)
```

Description

Returns the contents of the grid as a two dimensional array. The outer dimension is rows, the inner dimensions is columns.

jsonSelected

Synopsis

```
/(!jsonSelected)
```

Description

Returns a list of row numbers in the grid that are selected. The rows are numbered started from zero.

jsonValues

Synopsis

```
/(!jsonValues)
```

Description

Returns a list of the values of the options in the combo box control, as a Javascript list.

processAction(String actionName [, Object arg])

Synopsis

```
processAction(String actionName [, Object arg] )
```

Description

Sends a [control action](#) with the specified name to the control. The action name can be matched the with `/(!Action)` control command. If the action argument `arg` is specified, it may be retrieved with the `/(!ActionArg)` control command.

setVar(String var, Object val)

Synopsis

```
setVar(String var, Object val)
```

Description

Sets the value of the specified bean variable in the current control to `val`.

value

Synopsis

```
/(!value)
```

Description

Expands to the text value of the option that is currently selected.