Yet another EVS Valuehelp - Showing Displaytexts for Keys

By Bertram Ganz

Summary

In this article we demonstrate how to combine the advantages of the *Simple* as well as the *Extended Value Selector* in Web Dynpro applications. Whereas the SVS shows displaytexts instead of keys the EVS provides filtering and sorting functionality for finding keys in large valuesets. After having selected a key-displaytext-pair inside the generic EVS-UI the input field only displays the key stored in the underlying context attribute. The displaytext is not visible by default.

For eliminating this drawback we define a new *calculated context attribute* with the property *readOnly=true*. After having selected a key-displaytext-pair in the EVS-UI the user can immediately read the calculated displaytext in a *TextView*-UI-element next to the input field containing the underlying key value. This solution is called *EVS+* here.

In another variant (called *EVS++*) the input field itself is bound to a calculated context attribute of type *simple type* (containing an enumeration of key-displaytext-pairs) with the property *readOnly = false*. With this approach the displaytext can be displayed in an input field without losing the EVS functionality. Power users can still enter valid keys inside the input field without opening the selection popup.

Contents

Keywords

Web Dynpro valuehelp, Extended Value Selector, Simple Value Selector, calculated context attributes, calculated context attributes with readOnly=false, generic validation, displaytext, key, simple datatypes, validating actions, non-validating actions, Java Dictionary, Web Dynpro phase model

Acknowledgement

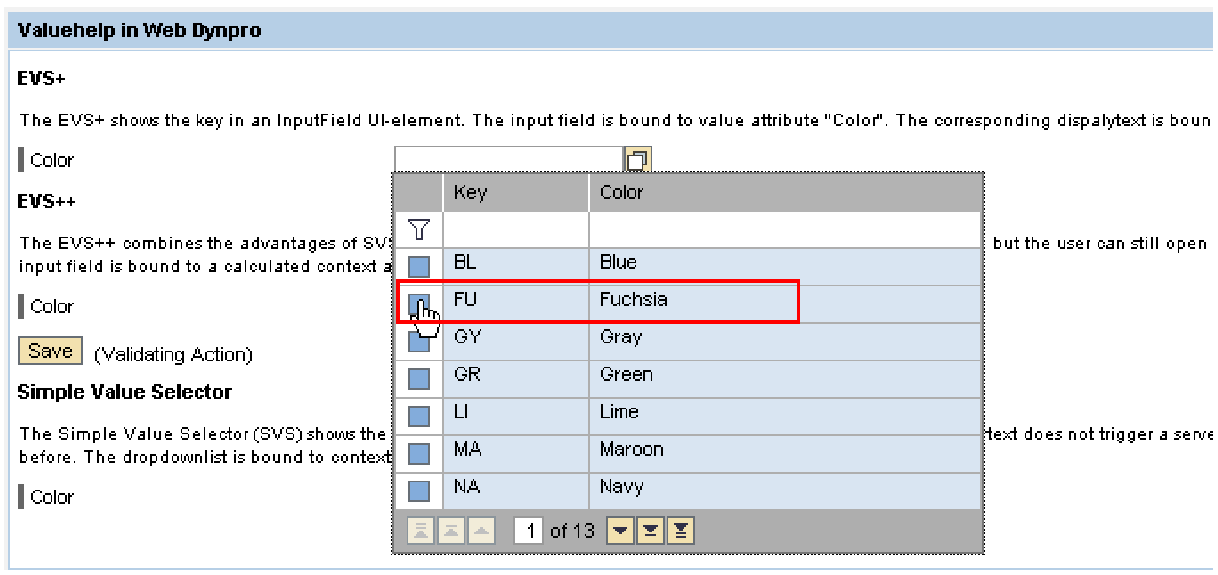
Many thanks to Markus Cherdron, Reiner Hammerich, Uwe Reeder and Jörg Singler for their valuable feedback and their helpful suggestions which are implied in the described EVS variants.

1. Introduction

A usability-drawback of the *Extended Value Selector* (EVS) is based on the fact, that only the key is displayed within the input field, but no displaytext. In contrast the *Simple Value Selector* (SVS) displays the displaytext in a *DropDownByKey*-UI-element without providing the EVS search and filtering functionality for large valuesets.
A combination of both advantages can be achieved by utilizing a **calculated context attribute**. Depending on the *readOnly*-property of this caculated context attribute two enhanced variants of the EVS-valuehelp can be implemented (called EVS+ and EVS++).

The following screenshots illustrate the behavior of the enhanced EVS-variants.

1. The UI of the enhanced EVS-variants is the same as the normal EVS UI. It provides sorting and simple filtering functionality (key, displaytext start with entered substring).



Valuehelp in Web Dynpro

EVS+

The EVS+ shows the key in an InputField UI-element. The input field is bound to value attribute "Color". The corresponding dispalytext is boun

Color

EVS++

The EVS++ combines the advantages of SVS … but the user can still open
input field is bound to a calculated context a

Color

Save (Validating Action)

Simple Value Selector

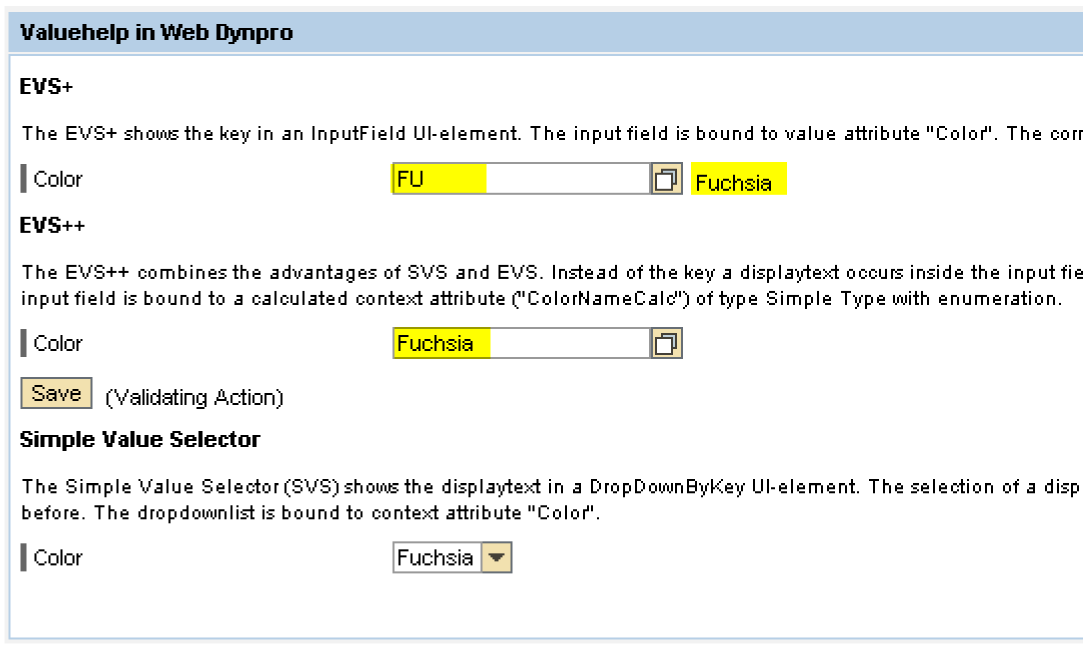
The Simple Value Selector (SVS) shows the … text does not trigger a serve
before. The dropdownlist is bound to context

Color

	Key	Color
	BL	Blue
	FU	Fuchsia
	GY	Gray
	GR	Green
	LI	Lime
	MA	Maroon
	NA	Navy

1 of 13

2. After having selected a key-displaytext-pair inside the EVS-UI the calculated **displaytext** is showing up next to the input field (variant **EVS+**).
In the **EVS++** variant this displaytext even appears inside the input field itself. This can be achieved by binding the input field to a calculated context attribute of type *readOnly=false* having the same simple datatype like the context attribute storing the key value.



Valuehelp in Web Dynpro

EVS+

The EVS+ shows the key in an InputField UI-element. The input field is bound to value attribute "Color". The corr

Color FU Fuchsia

EVS++

The EVS++ combines the advantages of SVS and EVS. Instead of the key a displaytext occurs inside the input fie
input field is bound to a calculated context attribute ("ColorNameCalc") of type Simple Type with enumeration.

Color Fuchsia

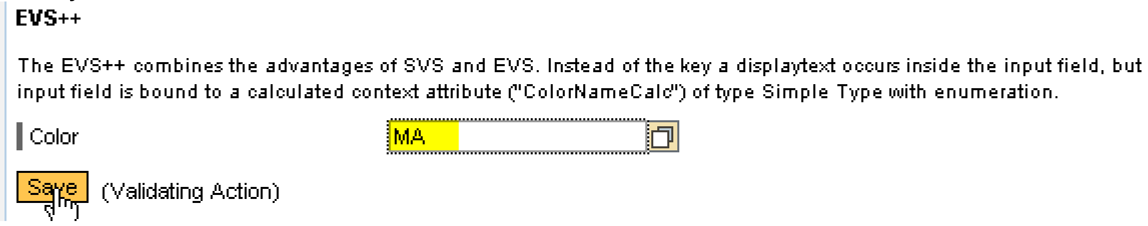
Save (Validating Action)

Simple Value Selector

The Simple Value Selector (SVS) shows the displaytext in a DropDownByKey UI-element. The selection of a disp
before. The dropdownlist is bound to context attribute "Color".

Color Fuchsia

3. The user can still enter a valid key in the input field for editing the context attribute, which stores the key value.



EVS++

The EVS++ combines the advantages of SVS and EVS. Instead of the key a displaytext occurs inside the input field, but
input field is bound to a calculated context attribute ("ColorNameCalc") of type Simple Type with enumeration.

Color MA

Save (Validating Action)

Displaytexts can only be entered when defining a *non-validating* action. Besides the given disadvantages of non-validating actions (no generic validation) the calculation of the correct key is not well-defined. Several keys with the same displaytext might exist.

2. Showing up displaytexts beside an EVS input field (EVS+)

We want to provide a valuehelp for the enumeration of 16 key-displaytext-pairs contained in a simple dictionary type named *Color*.

The valuehelps *EVS* and *SVS* are both bound to a context attribute *Color* of simple type *Color*. The context attribute stores the key-value. Whereas the SVS (*DropDownByKey*) displays the displaytext for a stored key, the EVS (*InputField*) displays the key directly. The major disadvantage of the EVS is based on the fact, that the corresponding displaytext is not visible for the user.

The ***EVS+* variant** is based on introducing a new **calculated context attribute** *ColorNameCalc* of type *Color* with the *readOnly*-property set to *true*. In the generated *getter*-method the displaytext for a given key is calculated. The calculated context attribute is only used for UI-purposes, so that the user can read the displaytext in a *TextView*-UI-Element being bound to this attribute.

It is important to know that the getter-method of the calculated attribute is called by the Web Dynpro Java Runtime in the same request response cycle in which the key-selection in the EVS-UI is handled and in which the EVS popup is closed. Allthough there is no chance for an application developer to get an action event handler being processed by the Web Dynpro Runtime he can use another hook: the *calculated attribute getter*. This getter is called by the Web Dynpro Runtime because a UI element in a visible view is bound to it so that the context data has to be retrieved.

Defining a calculated context attribute

First we define a new calculated context attribute with name **ColorNameCalc**.

- Set the *calculated*-property as *true*
- Set the *readOnly*-property as *true*
- The *type*-property (=*string*) must not be changed.

Adding a *TextView*-UI-element next to the input field

The *EVS+* is based on an additional *TextView*-UI-element next to the *InputField*-UI-element. For keeping both UI-elements together we put them into a *TransparentContainer*-UI-element.

- Add a new *TransparentContainer*-UI-element with *layout=FlowLayout*.
- Add a new *InputField*-UI-element to the transparent container and bind the *value*-property to the context attribute *Color*.
- Add a new *TextView*-UI-element to the same transparent container.
- Bind the *text*-property of the *TextView*-UI-element to the calculated context attribute *ColorNameCalc*.

Implementing the context calculation

The generated **getter-method** is called by the Web Dynpro Runtime for retrieving the calculated context attribute value during response rendering or when controller code accesses this attribute. The following steps have to be implemented:

- Access the `IWDAttributeInfo`-API of the context attribute *Color.*
- Access the `ISimpleValueSet`-API provided by the valueset contained in the context attribute's *Color* simple data type.
- Check wether the key stored in context attribute *Color* is valid.
- Return the displaytext for a valid key otherwise return an empty string.

```
  //@@begin javadoc:getColorNameCalc(IPrivateMain.IContextElement)
  /**
   *  Declared getter method for attribute MoreColorNameCalc of node Context
   *  @param element the element requested for the value
   *  @return the calculated value for attribute MoreColorNameCalc
   */
  //@@end
  public java.lang.String getColorNameCalc(IPrivateMain.IContextElement element) {
    //@@begin getColorNameCalc(IPrivateMain.IContextElement)
    String attributeName = IPrivateMain.IContextElement.COLOR;
    IWDAttributeInfo attributeInfo =
      element.node().getNodeInfo().getAttribute(attributeName);
    ISimpleType simpleType = attributeInfo.getSimpleType();
    ISimpleValueSet valueset = simpleType.getSVServices().getValues();
    Object key = element.getAttributeValue(attributeName);
    try {
      simpleType.checkValid(key);
      return valueset.getText(key);
    } catch (DdCheckException e) {
      return "";
    }
    //@@end
  }
```

3. Showing up displaytexts inside an EVS input field (EVS++)

We now want to show up displaytexts **inside** the input field itself – instead of keys (**variant *EVS++***, the additional *TextView*-UI-element in variant EVS+ can be omitted here). For achieving this the calculated context attribute *ColorNameCalc* has to be defined in a different way:

- the *readOnly*-property must be set to *false* instead of *true.*
- the context attribute *Color* and the calculated context attribute *ColorNameCalc* must have the same type (dictionary simple type containing an enumeration of key-value-pairs). Otherwise the EVS-UI cannot be opened from the input field.

Like in the EVS+ variant the calculated attribute is used for showing up the displaytext for the key value which is stored in the context. In contrast to the EVS and EVS+ the input field itself is bound to the calculated context attribute *ColorNameCalc* and not to the context attribute *Color* storing the key value. The calculated attribute is only used for UI-purposes, so that the user can read the displaytext within the input field. The selected key is still stored in context attribute *Color*.

But what's the purpose of the generated **setter-method** and when is it called? The synchronisation of both context attributes (*Color* and *ColorNameCalc*) is implemented within the generated getter- and setter-methodes called by the Web Dynpro Runtime after the generic validation of modified context data (setter-method) and during UI rendering (getter-method). During response rendering the *getter*-method is called by the Web Dynpro Runtime for retrieving the displaytext based on the value stored in context attribute *Color*. Also look at figure 1 and at figure 2 which visualizes the *Web Dynpro phase model*.

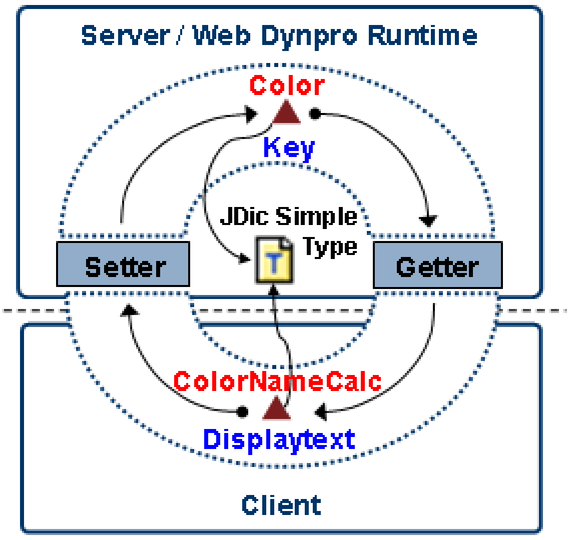


Figure 1. The EVS++ context

When a validating action is triggered on the UI the Web Dynpro Runtime first processes the generic validation phase before transporting modified data into the context. In our case the user first enters a valid key (*not* a displaytext) into the input field, e.g. *BL* for *Blue* before triggering a server roundtrip. Web Dynpro checks, whether this value is valid for the context attribute *ColorNameCalc*. Because *BL* is a valid key Web Dynpro then calls the setter-method `setDisplayText()` for calculating the context attribute *Color*. This attribute stores the key, based on the value displayed in the input field. Without implementing the setter-method the context attribute *Color* does not change.

The setter-method of a calculated context attribute is only called by the Web Dynpro Runtime when its value has changed on the UI and when this value is valid.

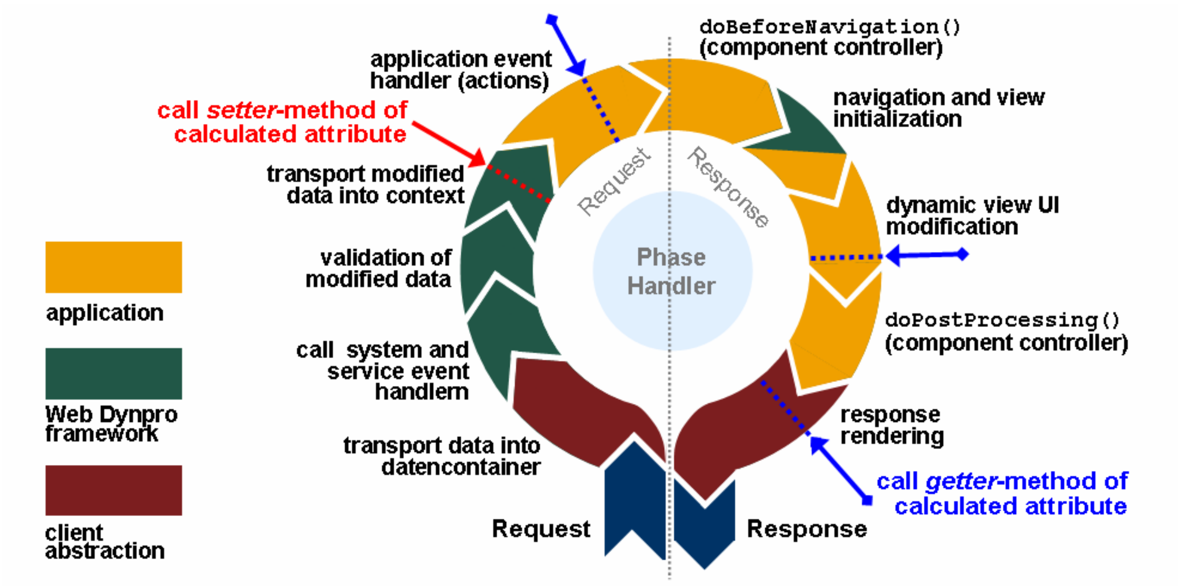

doBeforeNavigation()
(component controller)
application event handler (actions)
call *setter*-method of calculated attribute
navigation and view initialization
transport modified data into context
Request
Response
dynamic view UI modification
Phase Handler
validation of modified data
application
doPostProcessing()
(component controller)
call system and service event handlern
Web Dynpro framework
response rendering
transport data into datencontainer
call *getter*-method of calculated attribute
client abstraction
Request
Response

Figure 1. Web Dynpro phase model and calling context-calculation methods

Restrictions

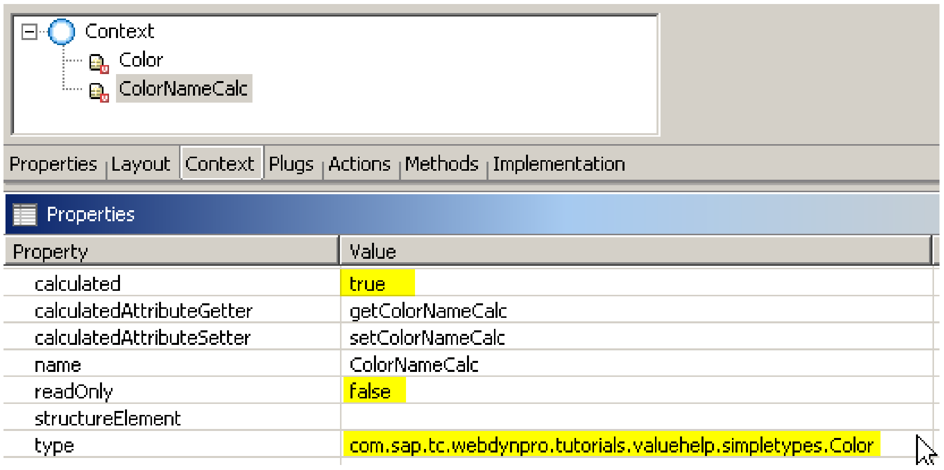
The described EVS++ variant is based on the fact, that the user enters valid keys in the input field. Invalid keys (like displaytexts differing from the keys) are rejected during the generic validation phase. This validation phase is processed by the Web Dynpro Runtime for all validating actions. It is only omitted for non-validating actions. In this case the user could also enter a displaytext (invalid key) so that the corresponding key can be determined in the setter-method. But this inverse operation (referring to the calculation of a displaytext for a given key) must not be unique. Think of an enumeration of key-displaytext-pairs containing the same displaytexts for different keys.

Users who want to insert displaytexts instead of keys simply make use of the EVS user interface.

Defining a calculated context attribute

First we change the declaration of the calculated context attribute **ColorNameCalc**.

- Set the *readOnly*-property as *false* (with *calculated= true*)
- For the *type*-property select the same dictionary simple type like for the context attribute *Color*. Otherwise the EVS-UI cannot be opened from the input field.



Property	Value
calculated	true
calculatedAttributeGetter	getColorNameCalc
calculatedAttributeSetter	setColorNameCalc
name	ColorNameCalc
readOnly	false
structureElement	
type	com.sap.tc.webdynpro.tutorials.valuehelp.simpletypes.Color

Adding an InputField-UI-element to the view layout

The *EVS*++is based on an *InputField*-UI-element which is bound to the calculated context attribute defined before.

- Add a new InputField-UI-element to the view layout.
- Bind it’s property *value* to the calculated context attribute *ColorNameCalc*.

Implementing the context calculation

The **getter-method** for calculating the context-attribute *ColorNameCalc* does not change. It is the same like in the EVS+ variant.

The additionally generated **setter-method** contains only one single line of code. We just copy the key value entered by the user from the calculated attribute *ColorNameCalc* to the second context attribute *Color*, which actually stores the key value in the context.

```
//@@begin javadoc:setColorNameCalc(IPrivateMain.IContextElement, java.lang.String)
/**
 *  Declared setter method for attribute ColorNameCalc of node Context
 *  @param element the element to change the value
 *  @param value the new value for attribute ColorNameCalc
 */
//@@end
public void setColorNameCalc(IPrivateMain.IContextElement element,
  java.lang.String value) {
  //@@begin setColorNameCalc(IPrivateMain.IContextElement, java.lang.String)

  // Precondition: the roundtrip is based on a validating action.
  // For non- validating actions the retrieved value may be invalid.
  element.setColor(value);

  //@@end
}
```

This implementation is based on the fact that a generic validation phase was processed by the Web Dynpro Runtime before, so that no invalid key can be passed to the *setColorNameCalc*-method. Remember that the context attribute *ColorNameCalc* has the datatype *Color*. Consequently the Web Dynpro Runtime checks, whether the entered value is valid or not. This validation phase is only omitted for non-validating actions.

For validating actions the setter-method is only called when the calculated context attribute *ColorNameCalc* stores (or the user inserted) a valid key. Consequently this value can directly be copied to the context attribute *Color* storing the key.

3. Conclusion

The described variants of the Extended Value Selector provide a mechanism for showing displaytexts for key-values stored in the context without losing the EVS valuehelp functionality. This can easily be achieved by defining an additional calculated context attribute. The generated getter- and setter-methods are called by the Web Dynpro Runtime for synchronizing the value attribute storing the key with the value attribute storing the displaytext and vice versa. In a more simple variant (with a calculated context attribute of type *readOnly=true*) the displaytext is just showing up beside the input field.