

1 Value Help – User Defined Programming

The Web Dynpro ABAP component model is one of the most powerful features of the environment. Component reuse and interoperability is the cornerstone of the ability to create your own custom value help components.

Although the Web Dynpro ABAP environment already has incredibly good support for existing Search Helps, you may find yourself wanting to create a more complex user interface for the search than can be done with Search Helps. Luckily there is also the option to embed our own custom Web Dynpro components via the User-Defined Programming option for the Input Help Mode of a Context Attribute.

In the following pages we will construct just such a custom Web Dynpro Value Help component. We will recreate the classic ABAP Dynpro time value help. Our new component will offer the Hour, Minutes, and Seconds as separate *DropDownByKey* UI elements. In addition we will support both a 24 hour mode and separate 12 hour mode with an *AM/PM RadioButton* element. The Clock button in the middle of the toolbar will switch between 24 and 12 hour modes.

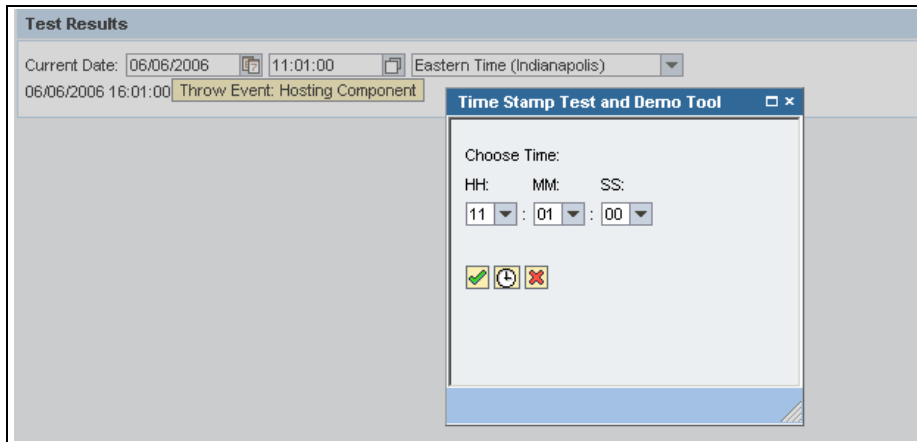


Figure 1.1: Example Help Component 24 Hour View

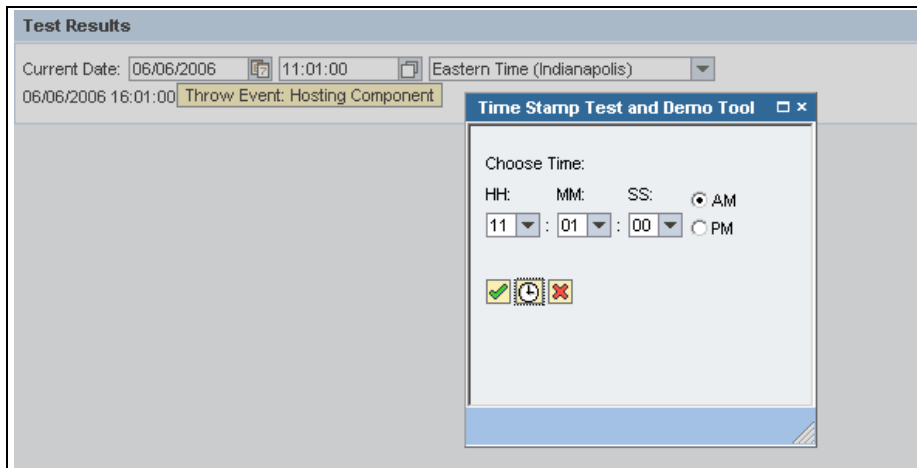


Figure 1.2: Example Help Component 12 Hour View

1.1 The Value Help Component

We will start our project by creating a new Web Dynpro Component just like normal. Once the component is created we need to add an interface to our component. This is the interface that will make this component compatible with the Value Help Processor. We can accomplish this by going to the Implemented Interfaces tab of the Component in change mode and adding the `IWD_VALUE_HELP` interface.

At first the Implementation column will show a red light. Press the button in the action column and the interface will be implemented within your component. Basically this will add some events and methods to your component control. We will work with these items later.

While we are in the component maintenance screen it is a good time to go ahead and generate an assistance class for this component. The assistance class will hold text elements, which can easily be translated into other languages, for use within the ABAP coding of our component.

If you type in the name of a global ABAP class that does not yet exist and press enter, the workbench will generate the class for you. In case you want to create the class yourself, you just need to remember to set the inheritance on the class. The super class should be `CL_WD_COMPONENT_ASSISTANCE`.

To finish off our assistance class, we need to go into change mode on the class itself. You can navigate to the assistance class from the Web Dynpro component by double clicking on it. We will need to add one text element that will be used later to create an error message. Once in edit mode in the global class builder, choose Goto->Text Elements. You can create a single text element with the ID E01 and the text “Data type is not appropriate for this value help object”. We will use this error message later if someone tries to use our value help component with a context attribute that is not a Time data type.

Once completed with these steps, your component should look like Figure 1.3.

Web Dynpro Component	Z_TIME_VALUE_HELP	Active
Description	Programmed Value Help for Time	
Assistance Class	ZCL_WDA_ASSIST_TIME_VALUE_HELP	
Created By	BCUSER	Created On 03/17/2006
Last Changed By	BCUSER	Changed On 06/06/2006
Package	ZE_BC_WDA_DEMOS	AccessibilityChecks Active <input checked="" type="checkbox"/>

Used Components	Implemented interfaces
-----------------	------------------------

Implemented Web Dynpro Component Interfaces			
Name	Description	Implemen..	Action
IWD_VALUE_HELP		○○●	

Figure 1.3: Component Maintenance Screen

1.2 Component Controller

We will start our component design with the component controller. Although our component will be fairly simple overall, it still is a good decision to build a robust component controller. So although we will eventually only have one view, we will go ahead and build our data context in our controller and map it to the view context later in the process.

Besides a good design practice, we have an added incentive to model our component this way when building a custom value help. The interaction with the value help processor is all done within the component controller. When we implemented the `IWD_VALUE_HELP` interface in our component, our component controller was the object that had events and methods added to it. In the end building our context at the component controller level also assist with the interaction to the value help processor.

1.2.1 Component Controller Context

Our component controller context will have a single node. This node has the following properties.

Property	Value
Node Name	INTERNAL_NODE
Interface Node	Not Checked
Input Element (Ext.)	Not Checked
Dictionary Structure	
Cardinality	1..1
Selection	0..1
Initialization Lead Selection	Checked
Singleton	Checked
Supply Function	

Component Controller Node Properties

Figure 1.4 shows the attributes that need to be created within our node.

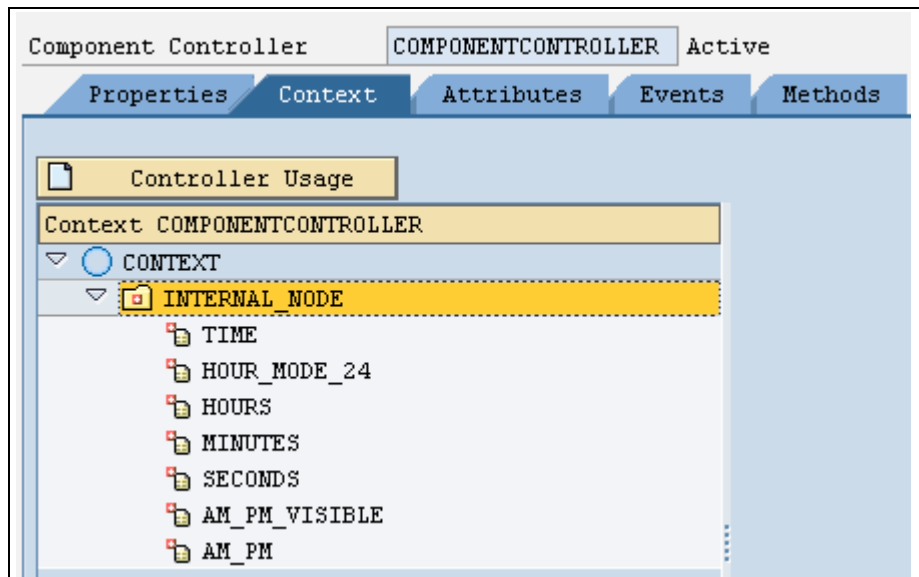


Figure 1.4: Component Controller Context Node and Attributes

We have an attribute named `TIME`. This will hold the time value in its internal ABAP format. We then have three attributes to hold the individual values that make up the time – `HOURS`, `MINUTES`, and `SECONDS`. Finally we have three attributes that are all related to the processing around 24/12 hour displays. We have the `HOUR_MODE_24` that controls which mode we are in. We have `AM_PM` where we store the current value – AM or PM- when in 12 hour mode. Finally we have the `AM_PM_VISIBLE` attribute that will later control if the AM/PM `RadioButton` group will be visible.

Attribute Name	Type
TIME	SYUZEIT
HOUR_MODE_24	WDY_BOOLEAN
HOURS	NUMC2

MINUTES	NUMC2
SECONDS	NUMC2
AM_PM_VISIBLE	WDUI_VISIBILITY
AM_PM	STRING

Component Controller Attribute Properties

1.2.2 Component Controller Attributes

In Figure 1.5 we can see the final attributes that we will want to have in our Component Controller. The first three attributes have all been generated for us; even the `WD_ASSIST` that points to an instance of the assistance class that we generated during the Component Controller maintenance.

Attribute	Publ.	RefTo	Associated Type	Description
WD_CONTEXT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	IF_WD_CONTEXT_NODE	Reference to Local Controller
WD_THIS	<input type="checkbox"/>	<input checked="" type="checkbox"/>	IF_COMPONENTCONTROLLER	Self-Reference to Local Conti
WD_ASSIST	<input type="checkbox"/>	<input checked="" type="checkbox"/>	ZCL_WDA_ASSIST_TIME_VALUE_HELP	Reference to the Instance of
VALUE_HELP_LISTENER	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	IF_WD_VALUE_HELP_LISTENER	

Figure 1.5: Component Controller Attributes

We will only need to add the fourth attribute. This attribute should be of `TYPE REF TO IF_WD_VALUE_HELP_LISTENER`. This is attribute will later be filled with a reference to an object that is part of the value help framework. This is the access to the context attribute that caused our value help component to be initiated. It is through this object that we view the data type of the source context attribute as well as interact with the context element. This is how we can both get and set the value in the originating context attribute!

It is worth taking a few minutes and studying the interface of the value help listener. First of all, it has only one method – `CLOSE_WINDOW`. At any time during the processing of our value help component if we want to close the popup window that we are displaying within, we can call this method of the value help listener. That is part of the reason that we will store a reference to the value help listener within the attributes of our component controller. This will give us easy access to the listener within any inner views we may have in our component.

The value help listener also provides a wealth of information about the source context item that triggered our value help in the form of its two attributes.

Attribute Name	Typing	Associated Type
F4_CONTEXT_ELEMENT	Type Ref To	IF_WD_CONTEXT_ELEMENT
F4_ATTRIBUTE_INFO	Type	WDR_CONTEXT_ATTRIBUTE_INFO

IF_WD_VALUE_HELP_LISTENER Attributes

The first attribute is rather self-explanatory. This is a reference to the originating Context Element itself. This gives us the full capabilities to interact with the originating context as though it was local to our component. Later in the methods of our component controller you will see the coding that we use to do this.

The second attribute provides plenty of good meta-data about our source context element. In this example we will only use the RTTI to read the source data type.

Component	Typing	Component Type
NAME	Type	STRING

DEFAULT_VALUE	Type	STRING
IS_READ_ONLY	Type	WDY_BOOLEAN
TYPE_NAME	Type	SABP_D_ABSOLUTE_TYPE_NAME
NODE_INFO	Type Ref To	IF_WD_CONTEXT_NODE_INFO
VALUE_SET	Type	WDR_CONTEXT_ATTR_VALUE_LIST
TYPE_ID	Type	INT4
RTTI	Type Ref To	CL_ABAP_DATADESCR
IS_STATIC	Type	WDY_BOOLEAN
IS_PRIMARY	Type	WDY_BOOLEAN
IS_NULLABLE	Type	WDY_BOOLEAN
REFERENCE_FIELD	Type	STRING
REFERENCE_FIELD_TYPE	Type	CHAR 1
VALUE_HELP_ID	Type	WDY_VALUE_HELP_ID
VALUE_HELP_MODE	Type	WDY_MD_VALUE_HELP_MODE_ENUM
VALUE_HELP_AVAILABLE	Type	WDY_VALUE_HELP_AVAILABLE
VALUE_HELP_TYPE	Type	SYCHAR01

WDR_CONTEXT_ATTRIBUTE_INFO Structure

1.2.3 Component Controller Events

When we implemented the `IWD_VALUE_HELP` we also received two events – `VH_WINDOW_CLOSED` and `VH_WINDOW_OPENED`. If we choose to set the interface attribute on these events, they can be exposed through the interface controller of our component as well.

In this example we will demonstrate how we can also raise custom events to the calling component. Therefore we will define a new event called `SELECTED` and set the interface attribute. We will raise this event when the user confirms a new value within our value help. That way the hosting component can perform processing on the change of the value via the value help.

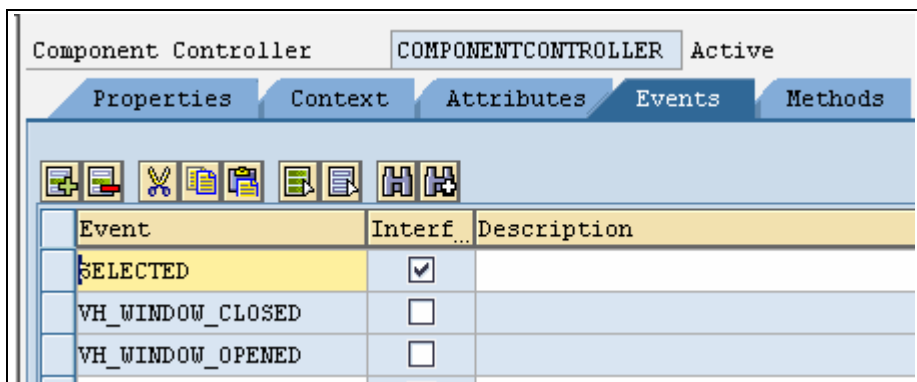


Figure 1.6: Component Controller Events

1.2.4 Component Controller Methods

Figure 1.7 shows the complete listing of methods that we will have in our component controller. We have all our standard Web Dynpro methods (`WDDOAPPLICATIONSTATECHANGE`, `WDDOBEFORENAVIGATION`, `WDDOEXIT`, `WDDO-`

`INIT`, and `WDDOPOSTPROCESSING`), although for this example we will only be implementing coding with `WDDOINIT`.

You will also notice the method `SET_VALUE_HELP_LISTENER`. This method was inserted when we implemented the `IWD_VALUE_HELP` Interface. This method will be called automatically each time that value help is requested. This is our opportunity to get our reference to the Value Help Listener and perform any initial validation or processing on the source context.

Finally we will add two custom methods to our controller. `SELECT` will be used to set a new value back into the source context. The second method, `SET_DDLB_VALUES`, will be used to populate the value sets for our `HOURS`, `MINUTES`, and `SECONDS` context attributes. These value sets will later be used by the `DropDown` UI elements.

Component Controller COMPONENTCONTROLLER Active

Properties Context Attributes Events **Methods**

Method	Method Ty...	Interf...	Description
SELECT	0 Method	<input type="checkbox"/>	Select the Entry
SET_DDLE_VALUES	0 Method	<input type="checkbox"/>	
SET_VALUE_HELP_LISTENER	0 Method	<input type="checkbox"/>	
WDDOAPPLICATIONSTATE	0 Method	<input type="checkbox"/>	Handling for Suspensi
WDDOBEFORENAVIGATION	0 Method	<input type="checkbox"/>	Error Handling Before
WDDOEXIT	0 Method	<input type="checkbox"/>	Cleanup Method of Cor
WDDOINIT	0 Method	<input type="checkbox"/>	Initialization Method
WDDOPOSTPROCESSING	0 Method	<input type="checkbox"/>	Prepare Output

Figure 1.7: Component Controller Methods

WDDOINIT

This will be our first method executed. Therefore it occurs before the `SET_VALUE_HELP_LISTENER` method, meaning that we do not have access to the source context yet. Because we can't get to the value help listener yet we will only use this method to setup the default values for the 24/12 hour mode.

If the same value help component is requested multiple times within the same application session, the `WDDOINIT` will only be fired once; whereas the `SET_VALUE_HELP_LISTENER` will fire at the beginning of each new value help request. This allows us to set the default value for the 4/12 hour mode once and if the user changes it through the interface that new mode will be remembered for the rest of their application session.

```

method wddoinit .

data:  node_internal_node  type ref to if_wd_context_node,
       elem_internal_node  type ref to if_wd_context_element,
       stru_internal_node  type if_componentcontroller=>element_internal_node .

* navigate from <CONTEXT> to <INTERNAL_NODE> via lead select
  node_internal_node =
    wd_context->get_child_node(
      name = if_componentcontroller=>wdctx_internal_node ).

* get element via lead selection
  elem_internal_node = node_internal_node->get_element( ).

* get all declared attributes
  elem_internal_node->get_static_attributes(
    importing
    static_attributes = stru_internal_node ).

  stru_internal_node-hour_mode_24 = abap_true.
  stru_internal_node-am_pm_visible =
    cl_wdl_core=>if_wdl_core~visibility_none.

* set all declared attributes
  elem_internal_node->set_static_attributes(
    exporting
    static_attributes = stru_internal_node ).

endmethod.

```

SET_VALUE_HELP_LISTENER

The main purpose of this method is to give our component the opportunity to access the value help listener. Therefore it has the following single parameter.

Parameter	Type	Ref To	Associated Type
LISTENER	Importing	Checked	IF_WD_VALUE_HELP_LISTENER

SET_VALUE_HELP_LISTENER Parameter listing

We wanted to save away the reference to the value help listener into an attribute of our component controller. Therefore we start the coding here.

```
method set_value_help_listener .
    wd_this->value_help_listener = listener.
```

Next we want to check the source data type and make sure that it is a date field. We will use the value help listener attribute `F4_ATTRIBUTE_INFO` to access the `RTTI` object of the source context element. If the data type is not a date, this is a serious error and we want to throw a fatal error. Notice that we will use our assistance class and the text element that we created earlier for the text of our error message.

```
    if wd_this->value_help_listener->f4_attribute_info-rtti-
>type_kind ne 'T'.

        data: error type string.
        error = wd_assist->if_wd_component_assistance~get_text(
            key = 'E01' ).
*        get message manager
        data: l_current_controller type ref to if_wd_controller,
            l_message_manager type ref to if_wd_message_manager.
        l_current_controller ?= wd_this->wd_get_api( ).
        l_message_manager =
            l_current_controller->get_message_manager( ).
*        report message
        l_message_manager->REPORT_FATAL_ERROR_MESSAGE(
```

```

        message_text = error ).
endif.

```

Next up we want to access the source context element and get the current value of the field that generated the value help request.

```

data: item_time type syzeit.
    wd_this->value_help_listener->f4_context_element-
>get_attribute(
    exporting name =
        wd_this->value_help_listener->f4_attribute_info-name
    importing value = item_time ).

```

Next we will want to take any initial value that we read out of the source context and copy it into our local context. We will also trigger the generation of the attribute value lists for the `DropDown` elements via the call to `SET_DDLB_VALUES`.

```

data:
    node_internal_node type ref to if_wd_context_node,
    elem_internal_node type ref to if_wd_context_element,
    stru_internal_node type if_componentcontroller=>elemen
t_internal_node .
* navigate from <CONTEXT> to <INTERNAL_NODE> via lead select
    node_internal_node = wd_context->get_child_node(
        name = if_componentcontroller=>wdctx_internal_node ).
* get element via lead selection
    elem_internal_node = node_internal_node->get_element( ).
* get all declared attributes
    elem_internal_node->get_static_attributes(
        importing static_attributes = stru_internal_node ).
    stru_internal_node-time = item_time.
* set all declared attributes
    elem_internal_node->set_static_attributes(
        exporting static_attributes = stru_internal_node ).

```

```
me->set_ddlb_values( ).
endmethod.
```

SET_DDLB_VALUES

The next method in our component controller is concerned with populating the value sets that will later be used by our `DropDown` UI elements. It is a fairly easy process with the only real complication being the different value set that must be created for the `HOURS` attribute depending upon the state of the 24/12 hour mode.

```
METHOD set_ddlb_values.
  DATA: node_internal_node TYPE REF TO if_wd_context_node,
        elem_internal_node TYPE REF TO if_wd_context_element,
        stru_internal_node TYPE if_componentcontroller=>element_in
ternal_node,
        l_node_info TYPE REF TO if_wd_context_node_info,
        item_hour_mode_24 LIKE stru_internal_node-hour_mode_24.

* navigate from <CONTEXT> to <INTERNAL_NODE> via lead select
node_internal_node = wd_context->get_child_node(
  name = if_componentcontroller=>wdctx_internal_node ).
* get element via lead selection
elem_internal_node = node_internal_node->get_element( ).

* get single attribute
elem_internal_node->get_attribute(
  EXPORTING name = `HOUR_MODE_24`
  IMPORTING value = item_hour_mode_24 ).

* Fill the Time Zone Value Set
DATA: wd_valueset TYPE wdy_key_value_table,
      wd_value TYPE wdy_key_value.
```

```

l_node_info ?= node_internal_node->get_node_info( ).
DATA: hours TYPE numc2, minutes TYPE numc2.
minutes = 60.
IF item_hour_mode_24 = abap_true.
    hours = 25.
ELSE.
    hours = 13.
ENDIF.

data: l_num type numc2.
DO hours TIMES.
    l_num = sy-index - 1.
    wd_value-key    = l_num.
    wd_value-value = l_num.
    INSERT wd_value INTO TABLE wd_valueset.
ENDDO.
l_node_info->set_attribute_value_set(
    name = `HOURS` value_set = wd_valueset ).

CLEAR wd_valueset.
DO minutes TIMES.
    l_num = sy-INDEX - 1.
    wd_value-KEY    = l_num.
    wd_value-VALUE = l_num.
    INSERT wd_value INTO TABLE wd_valueset.
ENDDO.
l_node_info->set_attribute_value_set(
    name = `MINUTES` value_set = wd_valueset ).
l_node_info->set_attribute_value_set(
    name = `SECONDS` value_set = wd_valueset ).
ENDMETHOD.

```

SELECT

The final method in our component controller will be responsible for copying the selected value in our help component back into the source context.

We will start processing by reading the current values in context. Remember these are the individual hours, minutes, and seconds fields.

```
method select .

    data: node_internal_node type ref to if_wd_context_node,
          elem_internal_node type ref to if_wd_context_element,
          stru_internal_node type
          if_componentcontroller=>element_internal_node .
    node_internal_node = wd_context->get_child_node(
        name = if_componentcontroller=>wdctx_internal_node ).
    elem_internal_node = node_internal_node->get_element( ).
    elem_internal_node->get_static_attributes(
        importing static_attributes = stru_internal_node ).
    data: l_time type syzeit.
```

Next we need to adjust these separate fields and put them back into the internal ABAP time format. If our user interface is currently in the 12 hour mode, we need to also adjust the hours field back to a 24 hour based value.

```
if stru_internal_node-hour_mode_24 = abap_true.
    l_time+0(2) = stru_internal_node-hours.
else.
    if stru_internal_node-am_pm = 'PM'.
        if stru_internal_node-hours eq '12'.
            l_time+0(2) = stru_internal_node-hours.
        else.
            l_time+0(2) = stru_internal_node-hours + 12.
        endif.
endif.
```

```

else.
    if stru_internal_node-hours eq '12'.
        l_time+0(2) = '00'.
    else.
        l_time+0(2) = stru_internal_node-hours.
    endif.
endif.
endif.
endif.
l_time+2(2) = stru_internal_node-minutes.
l_time+4(2) = stru_internal_node-seconds.

```

Finally we will use the value help listener to return the new value to the source context.

```

wd_this->value_help_listener->f4_context_element-
>set_attribute(
    exporting name =
        wd_this->value_help_listener->f4_attribute_info-name
        value = l_time ).
endmethod.

```

1.3 Component View

Within our component we will create a single view where we will place all of our user interface logic.

1.3.1 View Context

We will start within our View Context Tab. For this View, all the data that we need was already defined within the Component Controller. There all we need to do is to map the context node from our component controller to our view context. Figure 1.8 shows what your view context should look like after the mapping is complete.

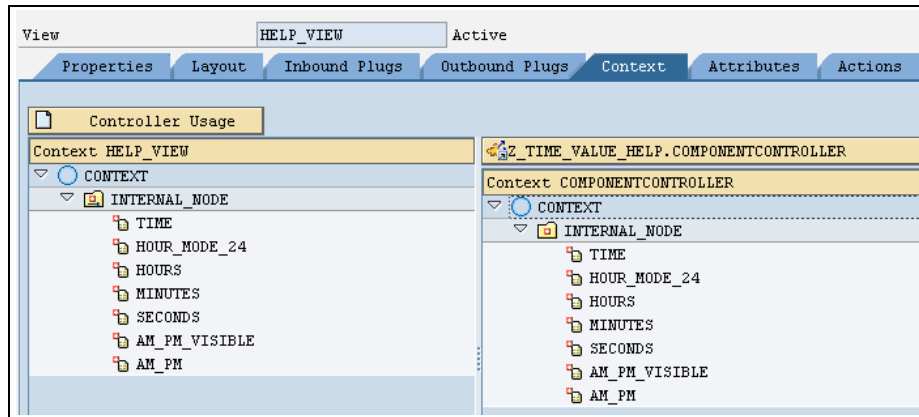


Figure 1.8: View Context

1.3.2 View Layout

Within the View Layout tab we model the look of our user interface. Figure 1.9 demonstrates the view layout preview as seen within SE80. Figure 1.10 shows the hierarchy of UI elements. `TransparentContainers` are used around each set of elements to assist in the positioning in the output. In this text we will list the properties of each UI element in the order that appear in the hierarchy.

Choose Time:

HH: MM: SS: AM

: : PM

Figure 1.9: View Layout Preview

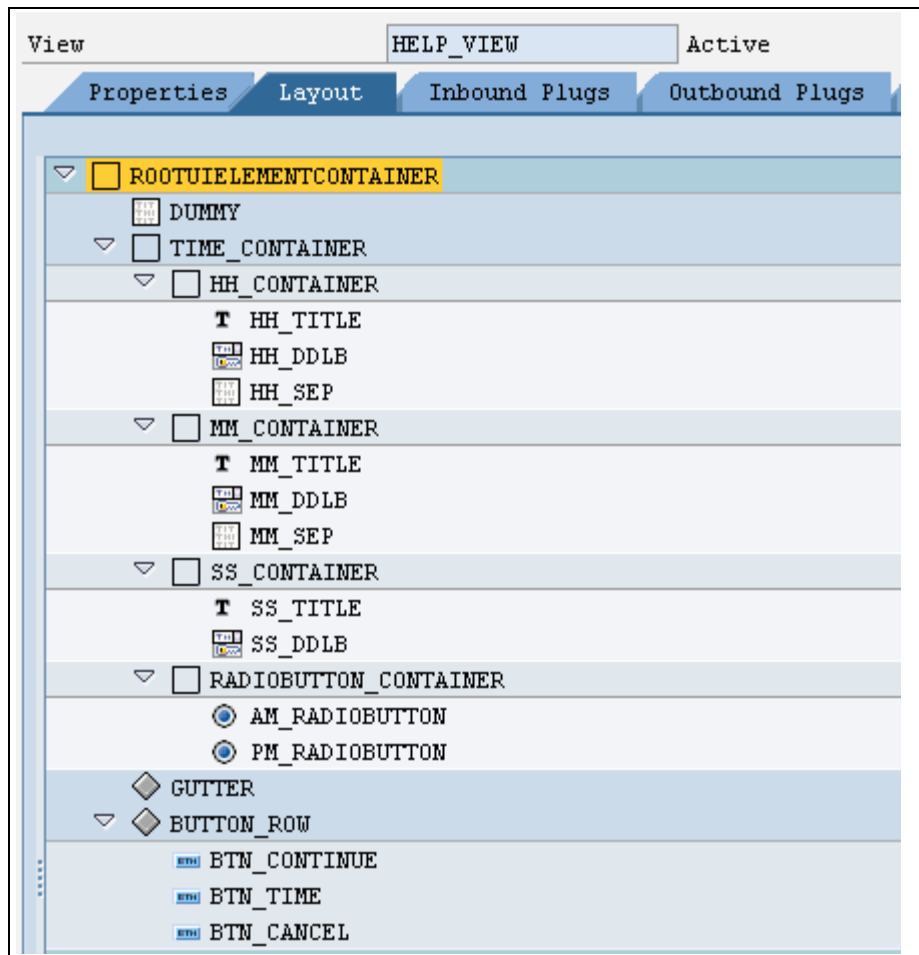


Figure 1.10: UI Element Hierarchy

Property	Value	Bin...
Properties (TransparentContainer)		
ID	ROOTUIELEMENTCONTAINER	
Layout	MatrixLayout	
accessibilityDesc...		
defaultButtonId		
enabled	<input checked="" type="checkbox"/>	
height	150px	
isLayoutContainer	<input checked="" type="checkbox"/>	
scrollingMode	none	
tooltip		
visible	Visible	
width	200px	
Layout (MatrixLayout)		
stretchedHorizont...	<input type="checkbox"/>	
stretchedVertically	<input type="checkbox"/>	

Figure 1.11: ROOTUIELEMENTCONTAINER

Property	Value	Bin...
Properties (TextView)		
ID	DUMMY	
Layout Data	MatrixHeadData	
design	standard	
enabled	<input checked="" type="checkbox"/>	
hAlign	auto	
layout	native	
semanticColor	standard	
text	Choose Time:	
textDirection	inherit	
tooltip		
visible	Visible	
wrapping	<input type="checkbox"/>	
Layout Data (MatrixHeadData)		
cellBackgroundDesign	transparent	
cellDesign	rPad	
colSpan	1	
height		
hAlign	beginOfLine	
vAlign	baseline	
vGutter	none	
width		

Figure 1.12: DUMMY – TextView

Property	Value	Binding
Properties (TransparentContainer)		
ID	TIME_CONTAINER	
Layout	GridLayout	
Layout Data	MatrixHeadData	
accessibilityDescrip...		
defaultButtonId		
enabled	<input checked="" type="checkbox"/>	
height		
isLayoutContainer	<input checked="" type="checkbox"/>	
scrollingMode	none	
tooltip		
visible	Visible	
width		
Layout (GridLayout)		
cellPadding	0	
cellSpacing	0	
colCount	4	
stretchedHorizontally	<input checked="" type="checkbox"/>	
stretchedVertically	<input checked="" type="checkbox"/>	
Layout Data (MatrixHeadData)		
cellBackgroundDesign	transparent	
cellDesign	rPad	
colSpan	1	
height		
hAlign	beginOfLine	
vAlign	Top	
vGutter	none	
width		

Figure 1.13: TIME_CONTAINER – TransparentContainer

Property	Value	Binding
Properties (TransparentContainer)		
ID	HH_CONTAINER	
Layout	MatrixLayout	
accessibilityDescrip...		
defaultButtonId		
enabled	<input checked="" type="checkbox"/>	
height		
isLayoutContainer	<input checked="" type="checkbox"/>	
scrollingMode	none	
tooltip		
visible	Visible	
width		
Layout (MatrixLayout)		
stretchedHorizontally	<input checked="" type="checkbox"/>	
stretchedVertically	<input checked="" type="checkbox"/>	
Layout Data (GridData)		
cellBackgroundDesign	transparent	
colSpan	1	
height		
hAlign	beginOfLine	
paddingBottom		
paddingLeft		
paddingRight		
paddingTop		
vAlign	baseline	
width		

Figure 1.14: HH_CONTAINER – TransparentContainer

Property	Value	Binding
Properties (Label)		
ID	HH_TITLE	
Layout Data	MatrixHeadData	
design	standard	
enabled	<input checked="" type="checkbox"/>	
labelFor	HH_DDLB	
text	HH	
textDirection	inherit	
tooltip		
visible	Visible	
width		
wrapping	<input type="checkbox"/>	
Layout Data (MatrixHeadData)		
cellBackgroundDesign	transparent	
cellDesign	rPad	
colSpan	1	
height		
hAlign	beginOfLine	
vAlign	baseline	
vGutter	none	
width		

Figure 1.15: HH_TITLE – Label

Property	Value	Binding
Properties (DropDownByKey)		
ID	HH_DDLB	
Layout Data	MatrixHeadData	
enabled	<input checked="" type="checkbox"/>	
explanation		
labelFor		
readOnly	<input type="checkbox"/>	
selectedKey	HELP_VIEW.INTERNAL_NODE.HOURS	
state	Normal Item	
textDirection	inherit	
tooltip		
visible	Visible	
width	20px	
Events		
onSelect		
Layout Data (MatrixHeadData)		
cellBackgroundDesign	transparent	
cellDesign	rPad	
colSpan	1	
height		
hAlign	beginOfLine	
vAlign	baseline	
vGutter	none	
width		

Figure 1.16: HH_DDLB – DropDownByKey

Property	Value	Binding
Properties (TextView)		
ID	HH_SEP	
Layout Data	MatrixData	
design	standard	
enabled	<input checked="" type="checkbox"/>	
hAlign	auto	
layout	native	
semanticColor	standard	
text	:	
textDirection	inherit	
tooltip		
visible	Visible	
wrapping	<input type="checkbox"/>	
Layout Data (MatrixData)		
cellBackgroundDesign	transparent	
cellDesign	rPad	
colSpan	1	
height		
hAlign	beginOfLine	
vAlign	baseline	
vGutter	none	
width		

Figure 1.17: HH_SEP – TextView

Property	Value	Binding
Properties (TransparentContainer)		
ID	MM_CONTAINER	
Layout	MatrixLayout	
accessibilityDescrip...		
defaultButtonId		
enabled	<input checked="" type="checkbox"/>	
height		
isLayoutContainer	<input checked="" type="checkbox"/>	
scrollingMode	none	
tooltip		
visible	Visible	
width		
Layout (MatrixLayout)		
stretchedHorizontally	<input checked="" type="checkbox"/>	
stretchedVertically	<input checked="" type="checkbox"/>	
Layout Data (GridData)		
cellBackgroundDesign	transparent	
colSpan	1	
height		
hAlign	beginOfLine	
paddingBottom		
paddingLeft		
paddingRight		
paddingTop		
vAlign	baseline	
width		

Figure 1.18: MM_CONTAINER – TransparentContainer

Property	Value	Binding
Properties (Label)		
ID	MM_TITLE	
Layout Data	MatrixHeadData	
design	standard	
enabled	<input checked="" type="checkbox"/>	
labelFor	MM_DDLB	
text	MM	
textDirection	inherit	
tooltip		
visible	Visible	
width		
wrapping	<input type="checkbox"/>	
Layout Data (MatrixHeadData)		
cellBackgroundDesign	transparent	
cellDesign	rPad	
colSpan	1	
height		
hAlign	beginOfLine	
vAlign	baseline	
vGutter	none	
width		

Figure 1.19: MM_TITLE – Label

Property	Value	Binding
Properties (DropDownByKey)		
ID	MM_DDLB	
Layout Data	MatrixHeadData	
enabled	<input checked="" type="checkbox"/>	
explanation		
labelFor		
readOnly	<input type="checkbox"/>	
selectedKey	HELP_VIEW.INTERNAL_NODE.MINUTES	
state	Normal Item	
textDirection	inherit	
tooltip		
visible	Visible	
width	20px	
Events		
onSelect		
Layout Data (MatrixHeadData)		
cellBackgroundDesign	transparent	
cellDesign	rPad	
colSpan	1	
height		
hAlign	beginOfLine	
vAlign	baseline	
vGutter	none	
width		

Figure 1.20: MM_DDLB – DropDownByKey













Property	Value	Binding
Properties (TextView)		
ID	MM_SEP	
Layout Data	MatrixData 	
design	standard 	
enabled	<input checked="" type="checkbox"/>	
hAlign	auto 	
layout	native 	
semanticColor	standard 	
text	:	
textDirection	inherit 	
tooltip		
visible	Visible 	
wrapping	<input type="checkbox"/>	
Layout Data (MatrixData)		
cellBackgroundDesign	transparent 	
cellDesign	rPad 	
colSpan	1	
height		
hAlign	beginOfLine 	
vAlign	baseline 	
vGutter	none 	
width		

Figure 1.21: MM_SEP – TextView

Property	Value	Binding
Properties (TransparentContainer)		
ID	SS_CONTAINER	
Layout	MatrixLayout	
accessibilityDescrip...		
defaultButtonId		
enabled	<input checked="" type="checkbox"/>	
height		
isLayoutContainer	<input checked="" type="checkbox"/>	
scrollingMode	none	
tooltip		
visible	Visible	
width		
Layout (MatrixLayout)		
stretchedHorizontally	<input checked="" type="checkbox"/>	
stretchedVertically	<input checked="" type="checkbox"/>	
Layout Data (GridData)		
cellBackgroundDesign	transparent	
colSpan	1	
height		
hAlign	beginOfLine	
paddingBottom		
paddingLeft		
paddingRight		
paddingTop		
vAlign	baseline	
width		

Figure 1.22: SS_CONTAINER – TransparentContainer

Property	Value	Binding
Properties (Label)		
ID	SS_TITLE	
Layout Data	MatrixHeadData	
design	standard	
enabled	<input checked="" type="checkbox"/>	
labelFor	SS_DDLB	
text	SS	
textDirection	inherit	
tooltip		
visible	Visible	
width		
wrapping	<input type="checkbox"/>	
Layout Data (MatrixHeadData)		
cellBackgroundDesign	transparent	
cellDesign	rPad	
colSpan	1	
height		
hAlign	beginOfLine	
vAlign	baseline	
vGutter	none	
width		

Figure 1.23: SS_TITLE – Label














Property	Value	Binding
Properties (DropDownByKey)		
ID	SS_DDLB	
Layout Data	MatrixHeadData 	
enabled	<input checked="" type="checkbox"/>	
explanation		
labelFor		
readOnly	<input type="checkbox"/>	
selectedKey	HELP_VIEW.INTERNAL_MODE.SECONDS 	
state	Normal Item 	
textDirection	inherit 	
tooltip		
visible	Visible 	
width	20px	
Events		
onSelect		 
Layout Data (MatrixHeadData)		
cellBackgroundDesign	transparent 	
cellDesign	rPad 	
colSpan	1	
height		
hAlign	beginOfLine 	
vAlign	baseline 	
vGutter	none 	
width		

Figure 1.24: SS_DDLB – DropDownByKey


Property	Value	Binding
Properties (TransparentContainer)		
ID	RADIOBUTTON_CONTAINER	
Layout	GridLayout	
accessibil...		
defaultBut...		
enabled	<input checked="" type="checkbox"/>	
height		
isLayoutCo...	<input checked="" type="checkbox"/>	
scrollingMode	none	
tooltip		
visible	HELP_VIEW.INTERNAL_NODE.AM_PM_VISIBLE	
width		
Layout (GridLayout)		
stretchedH...	<input type="checkbox"/>	
stretchedV...	<input type="checkbox"/>	
Layout Data (GridData)		
cellBackgr...	transparent	
colSpan	1	
height		
hAlign	beginOfLine	
paddingBottom		
paddingLeft		
paddingRight		
paddingTop		
vAlign	baseline	
width		

Figure 1.25: RADIOBUTTON_CONTAINER – TransparentContainer












Property	Value	Binding
Properties (RadioButton)		
ID	AM_RADIOBUTTON	
Layout Data	MatrixHeadData 	
enabled	<input checked="" type="checkbox"/>	
explanation		
keyToSelect	AM	
readOnly	<input type="checkbox"/>	
selectedKey	HELP_VIEW.INTERNAL_NODE.AM_PM 	
state	Normal Item 	
text	AM	
textDirection	inherit 	
tooltip		
visible	Visible 	
Events		
onSelect		
Layout Data (MatrixHeadData)		
cellBackgroundDesign	transparent 	
cellDesign	rPad 	
colSpan	1	
height		
hAlign	beginOfLine 	
vAlign	baseline 	
vGutter	none 	
width		

Figure 1.26: AM_RADIOBUTTON – RadioButton













Property	Value	Binding
Properties (RadioButton)		
ID	PM_RADIOBUTTON	
Layout Data	MatrixHeadData 	
enabled	<input checked="" type="checkbox"/>	
explanation		
keyToSelect	PM	
readOnly	<input type="checkbox"/>	
selectedKey	HELP_VIEW.INTERNAL_MODE.AM_PM 	
state	Normal Item 	
text	PM	
textDirection	inherit 	
tooltip		
visible	Visible 	
Events		
onSelect		 
Layout Data (MatrixHeadData)		
cellBackgroundDesign	transparent 	
cellDesign	rPad 	
colSpan	1	
height		
hAlign	beginOfLine 	
vAlign	baseline 	
vGutter	none 	
width		

Figure 1.27: PM_RADIOBUTTON – RadioButton

Property	Value	Binding
Properties (InvisibleElement)		
ID	GUTTER	
Layout Data	MatrixHeadData	
enabled	<input checked="" type="checkbox"/>	
tooltip		
visible	Visible	
Layout Data (MatrixHeadData)		
cellBackgroundDesign	transparent	
cellDesign	rPad	
colSpan	1	
height	20px	
hAlign	beginOfLine	
vAlign	baseline	
vGutter	none	
width		

Figure 1.28: GUTTER – InvisibleElement








Property	Value	Binding
Properties (ButtonRow)		
ID	BUTTON_ROW	
Layout Data	MatrixHeadData 	
enabled	<input checked="" type="checkbox"/>	
tooltip		
visible	Visible 	
Layout Data (MatrixHeadData)		
cellBackgroundDesign	transparent 	
cellDesign	rPad 	
colSpan	1	
height		
hAlign	beginOfLine 	
vAlign	baseline 	
vGutter	none 	
width		

Figure 1.29: BUTTON_ROW – ButtonRow






Property	Value	Binding
Properties (Button)		
ID	BTN_CONTINUE	
design	standard	
enabled	<input checked="" type="checkbox"/>	
explanation		
imageFirst	<input checked="" type="checkbox"/>	
imageSource	WEBICON_CHECKED	
text		
textDirection	inherit	
tooltip		
visible	Visible	
width		
Events		
onAction	SELECT	 

Figure 1.30: BTN_CONTINUE – Button

Property	Value	Binding
<u>Properties (Button)</u>		
ID	BTN_TIME	
design	standard	
enabled	<input checked="" type="checkbox"/>	
explanation		
imageFirst	<input checked="" type="checkbox"/>	
imageSource	ICON_TIME	
text		
textDirection	inherit	
tooltip		
visible	Visible	
width		
<u>Events</u>		
onAction	CHANGE_TIME_MODE	

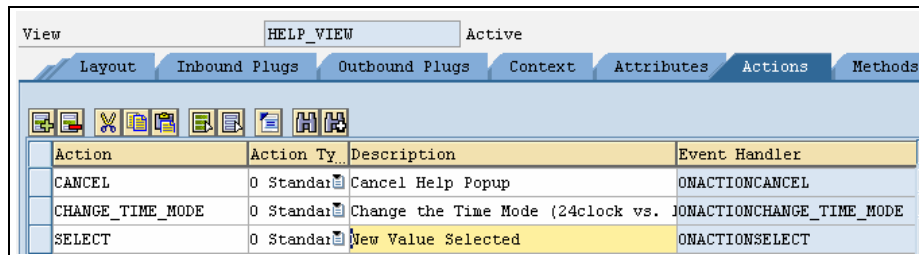
Figure 1.31: BTN_TIME – Button

Property	Value	Binding
<u>Properties (Button)</u>		
ID	BTN_CANCEL	
design	standard	
enabled	<input checked="" type="checkbox"/>	
explanation		
imageFirst	<input checked="" type="checkbox"/>	
imageSource	WEBICOM_CRITICAL	
text		
textDirection	inherit	
tooltip		
visible	Visible	
width		
<u>Events</u>		
onAction	CANCEL	

Figure 1.32: BTN_CANCEL – Button

1.3.3 View Actions

During the creation of our `Button` UI elements, there were different actions that were attached to the `onAction` events. Figure 1.33 shows these three actions. Each one is tied to one of the buttons near the bottom of our user interface. `CANCEL` will be used to exit the value help popup. `SELECT` will trigger the current value to be returned to the calling component. Finally `CHANGE_TIME_MODE` will cause the toggle between 24/12 hour modes.



Action	Action Ty...	Description	Event Handler
CANCEL	0 Standard	Cancel Help Popup	ONACTIONCANCEL
CHANGE_TIME_MODE	0 Standard	Change the Time Mode (24clock vs.	ONACTIONCHANGE_TIME_MODE
SELECT	0 Standard	New Value Selected	ONACTIONSELECT

Figure 1.33: View Actions

View Methods

In our listing of methods, Figure 1.34, we see all the standard methods – `WDDOBEFOREACTION`, `WDDOEXIT`, `WDDOINIT`, `WDDOMODIFYVIEW`. From this list we will use `WDDOBEFOREACTION` to before validations on the input values and `WDDOINIT` for view initialization.

We also have our three event handler methods that match up to our actions – `ONACTIONCANCEL`, `ONACTIONCHANGE_TIME_MODE`, and `ONACTIONSELECT`. Finally we will add one custom method, `INITIALIZE_AM_PM`, to help with the processing during a 24/12 hour mode switch.

View			
HELP_VIEW Active(revised)			
Layout Inbound Plugs Outbound Plugs Context Attributes Actions Methods			
[Icons]			
Method	Method Type	Description	Event
OMACTIONCANCEL	1 Event Handle	Cancel Help Popup	
OMACTIONCHANGE_TIME_MODE	1 Event Handle	Change the Time Mode (24clock vs.	
OMACTIONSELECT	1 Event Handle	New Value Selected	
INITIALIZE_AM_PM	0 Method	Initialize the AM/PM Setting	
WDDOBEFOREACTION	0 Method	Method for Validation of User Input	
WDDOEXIT	0 Method	Cleanup Method of Controller	
WDDOINIT	0 Method	Initialization Method of Controller	
WDDOMODIFYVIEW	0 Method	Method for Modifying the View Before	

Figure 1.34: View Methods

WDDOINIT

Our source data element should have the ABAP internal date format, but for our user interface we want to work with the hours, minutes and seconds a separate UI elements. Therefore during the view initialization we will split the current value into its separate components. We will also trigger a call to INITIALIZE_AM_PM in order adjust the hours based upon the current 24/12 hour mode setting.

```

METHOD wddoinit .
    DATA: node_internal_node TYPE REF TO if_wd_context_node,
           elem_internal_node TYPE REF TO if_wd_context_element,
           stru_internal_node
           TYPE if_help_view=>element_internal_node .
*   navigate from <CONTEXT> to <INTERNAL_NODE> via lead select
    node_internal_node = wd_context->get_child_node(
        name = if_help_view=>wdctx_internal_node ).
*   get element via lead selection
    elem_internal_node = node_internal_node->get_element( ).
*   get all declared attributes
    elem_internal_node->get_static_attributes(

```

```

    IMPORTING static_attributes = stru_internal_node ).

stru_internal_node-hours    = stru_internal_node-time+0(2).
stru_internal_node-minutes = stru_internal_node-time+2(2).
stru_internal_node-seconds = stru_internal_node-time+4(2).

elem_internal_node->set_static_attributes(
EXPORTING static_attributes = stru_internal_node ).
me->initialize_am_pm( abap_false ).
ENDMETHOD.

```

INITIALIZE_AM_PM

When switching between 24 and 12 hour modes, the hour's value obviously needs to be adjusted by +/- 12 hours. That is the purpose of this method. There is one special case however. If we already have a 24-based hour value internally and we are initialized to the 24 hour mode, we do not want to add another 12 hours to the current value.

Therefore we have a single parameter, `MODE_SWITCH`, which controls if the 12 hours are added on. During view initialization we set this `MODE_SWITCH` to false to avoid this extra addition.

Parameter	Type	Ref To	Associated Type
MODE_SWITCH	Importing		WDY_BOOLEAN

INITIALIZE_AM_PM Parameters

```

METHOD initialize_am_pm .
    DATA: node_internal_node TYPE REF TO if_wd_context_node,
           elem_internal_node TYPE REF TO if_wd_context_element,
           stru_internal_node TYPE if_help_view=>element_internal_node .
    *   navigate from <CONTEXT> to <INTERNAL_NODE> via lead select

```

```

node_internal_node = wd_context->get_child_node(
    name = if_help_view=>wdctx_internal_node ).
*   get element via lead selection
elem_internal_node = node_internal_node->get_element( ).
*   get all declared attributes
elem_internal_node->get_static_attributes(
IMPORTING static_attributes = stru_internal_node ).

IF stru_internal_node-hour_mode_24 = abap_false.
    stru_internal_node-am_pm_visible =
        cl_wdl_core=>if_wdl_core-visibility_visible.
    IF stru_internal_node-hours EQ '24'.
        stru_internal_node-am_pm = 'PM'.
        stru_internal_node-hours = '00'.
    ELSEIF stru_internal_node-hours EQ '00'.
        stru_internal_node-am_pm = 'AM'.
        stru_internal_node-hours = '12'.
    ELSEIF stru_internal_node-hours EQ '12'.
        stru_internal_node-am_pm = 'PM'.
        stru_internal_node-hours = '12'.
    ELSEIF stru_internal_node-hours GT '12'.
        stru_internal_node-am_pm = 'PM'.
        stru_internal_node-hours =
            stru_internal_node-hours - 12.
****Not needed after 04S SP7 Upgrade
*   if stru_internal_node-hours < 10.
*       stru_internal_node-hours+1(1) =
*           stru_internal_node-hours(1).
*       stru_internal_node-hours(1) = '0'.
*   endif.
    ELSE.
        stru_internal_node-am_pm = 'AM'.
    ENDIF.
ELSE.
    stru_internal_node-am_pm_visible =

```

```

        cl_wdl_core=>if_wdl_core~visibility_none.
    IF mode_switch = abap_true.
        IF stru_internal_node-am_pm = 'PM'.
            IF stru_internal_node-hours EQ '12'.
                ELSE.
                    stru_internal_node-hours =
                        stru_internal_node-hours + 12.
                ENDIF.
            ELSE.
                IF stru_internal_node-hours EQ '12'.
                    stru_internal_node-hours = '00'.
                ENDIF.
            ENDIF.
        ENDIF.
    ENDIF.
    ENDIF.
    ENDIF.
    elem_internal_node->set_static_attributes(
        EXPORTING static_attributes = stru_internal_node ).
ENDMETHOD.

```

WDDOBEFOREACTION

In this method we will perform our validations on the time value. Basically we just need to check to make sure that invalid times, like 00:00:00, produce an error.

We will start our processing by reading our context.

```

METHOD wddobeforeaction .
    DATA: node_internal_node TYPE REF TO if_wd_context_node,
           elem_internal_node TYPE REF TO if_wd_context_element,
           stru_internal_node TYPE if_help_view=>element_internal_node.
    *   navigate from <CONTEXT> to <INTERNAL_NODE> via lead select
        node_internal_node = wd_context->get_child_node(

```

```

        name = if_help_view=>wdctx_internal_node ).
*   get element via lead selection
    elem_internal_node = node_internal_node->get_element( ).
*   get all declared attributes
    elem_internal_node->get_static_attributes(
        IMPORTING static_attributes = stru_internal_node ).

```

Next we will request an instance of the message manager. This makes the coding for issuing error messages simpler later.

```

*   get message manager
DATA: l_current_controller TYPE REF TO if_wd_controller,
      l_message_manager   TYPE REF TO if_wd_message_manager.
l_current_controller ?= wd_this->wd_get_api( ).
l_message_manager =
    l_current_controller->get_message_manager( ).

```

Now we have our data validations. There are already standard error messages that meet the needs for our validations. To use them we will use the message manager method `REPORT_T100_MESSAGE`. We can then access classic Message Class messages.

```

IF stru_internal_node-hour_mode_24 EQ abap_true.
ELSE.
    IF stru_internal_node-am_pm EQ 'AM'.
        IF stru_internal_node-hours EQ '12'.
            stru_internal_node-hours = '00'.
        ELSEIF stru_internal_node-hours EQ '00'.
            IF stru_internal_node-minutes NE '00' OR
               stru_internal_node-seconds NE '00'.
                l_message_manager->report_t100_message(
                    msgid = 'SH' msgno = '010' msgty = 'E' ).
            ELSE.
                stru_internal_node-hours = '12'.
            ENDIF.
        ENDIF.
    ENDIF.

```

```

ELSE.
  IF stru_internal_node-hours EQ '12'.
  ELSEIF stru_internal_node-hours EQ '00'.
    IF stru_internal_node-minutes NE '00' OR
      stru_internal_node-seconds NE '00'.
      l_message_manager->report_t100_message(
        msgid = 'SH' msgno = '010' msgty = 'E' ).
    ELSE.
      stru_internal_node-hours = '24'.
    ENDIF.
  ENDIF.
ENDIF.
ENDIF.
ENDMETHOD.

```

ONACTIONCANCEL

This method only needs to tell the value help listener to close the popup window because the user has chosen to cancel processing. No value will be copied back to the source context.

```

METHOD onactioncancel .
  wd_comp_controller->value_help_listener->close_window( ).
ENDMETHOD.

```

ONACTIONCHANGE_TIME_MODE

This method will toggle the 24/12 hour mode. We start the processing by reading the current value of the attribute.

```

METHOD onactionchange_time_mode .
  DATA: node_internal_node TYPE REF TO if_wd_context_node,
        elem_internal_node TYPE REF TO if_wd_context_element,

```

```

stru_internal_node TYPE if_help_view=>element_internal_node.

node_internal_node = wd_context->get_child_node(
    name = if_help_view=>wdctx_internal_node ).
elem_internal_node = node_internal_node->get_element( ).
elem_internal_node->get_static_attributes(
    IMPORTING static_attributes = stru_internal_node ).

```

Next we will toggle the value itself. We then may need to reset the hour value based upon the mode. Finally we will reload the value sets for the `DropDown` elements.

```

IF stru_internal_node-hour_mode_24 = abap_true.
    stru_internal_node-hour_mode_24 = abap_false.
ELSE. stru_internal_node-hour_mode_24 = abap_true.
ENDIF.
elem_internal_node->set_static_attributes(
    EXPORTING static_attributes = stru_internal_node ).
me->initialize_am_pm( abap_true ).
wd_comp_controller->set_ddlb_values( ).
ENDMETHOD.

```

ONACTIONSELECT

Our final method is fired when the user wants to copy their current value from the value help back into the source context. The logic to copy the current value back into the source context is all contained in the component controller method `SELECT`. We will then ask the component controller to fire our interface event `SELECTED`. Finally we ask the value help listener to close the popup window.

```

METHOD onactionselect .
* move selected value to special context attribute
    wd_comp_controller->select( ).
* fire event: application component is now getting the
* selected value out of the special context attribute

```

```
wd_comp_controller->fire_selected_evt( ).
* close popup
wd_comp_controller->value_help_listener->close_window( ).
ENDMETHOD.
```

1.3.4 View Window Relationship

Although we only have a single view, the mapping of view to the window is a relatively uneventful activity. However don't forget this minor detail. Otherwise when you go to test the value help, you be left wondering why nothing is displayed in the popup window.

1.4 Using the Value Help Component

Now that we have completed our custom value help component we are ready to put it to use. The process is designed to be relatively unobtrusive to the hosting component.

If we create a small test component, we first need to add a component usage for our value help component as shown in Figure 1.35. The name that you give to the component usage does not really matter. Here we have given it the name `VALUE_HELP`.


Component Controller		COMPONENTCONTROLLER	Active
Properties		Context	Attributes
Events		Methods	
Description			
Created By	BCUSER	Created on	03/16/2006
Last changed by	BCUSER	Changed on	06/06/2006
			
Used Controllers/Components			
Component Use	Component	Controller	
	Z_TIME_STAMP	Z_TIME_STAMP	
VALUE_HELP	Z_TIME_VALUE_HELP		
VALUE_HELP	Z_TIME_VALUE_HELP	INTERFACECONTROLLER	

Figure 1.35: Component Usage Creation

Next we go to the context that contains the attribute we want to link to our User-Defined Programming based Value Help. We simply override the Input Help Mode Value on the Attribute. Once set to User-Defined Programming, an addition field will appear for the selection of the Component Usage we want to use.

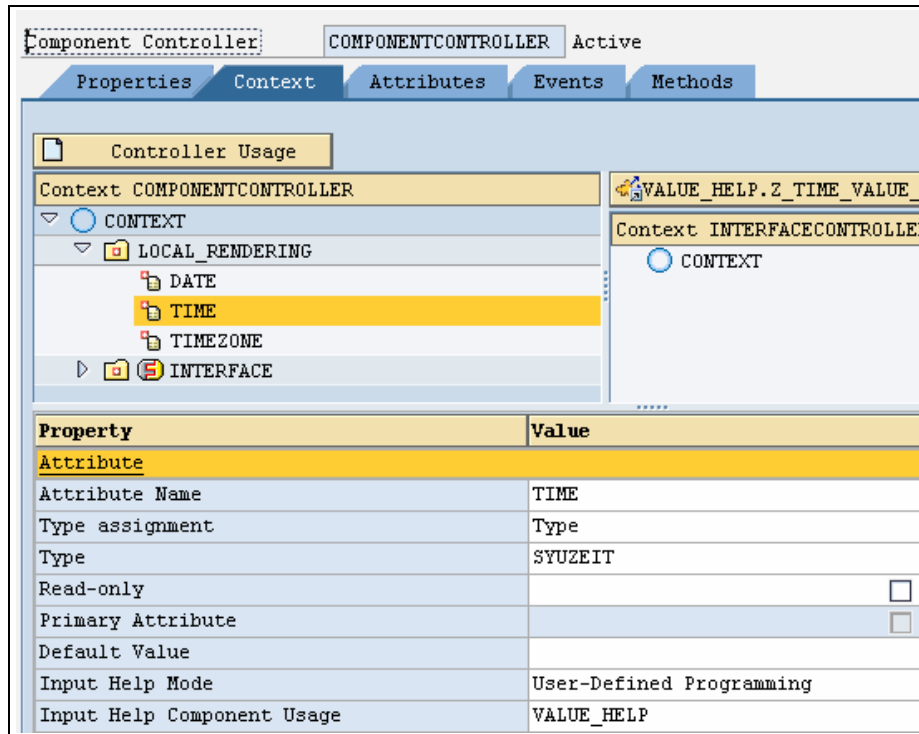


Figure 1.36: Assignment of our Component Usage to the Target Attribute

With that the connection between the hosting component and our custom value help component is complete. In this example application we have set this value on a context attribute within the component controller.

We later mapped this context to the view context and then used that attribute in an `InputField`. No changes were needed to the view to use our custom value help component. All the necessarily settings are inherited from the component controller context via the context mapping.