# New Features of Web Dynpro Popup Window - SAP NetWeaver CE 7.11

## Applies to:

Web Dynpro for Java (SAP NetWeaver CE 7.11). For more information, visit the User Interface Technology homepage.

## Summary

With SAP NetWeaver CE 7.11, a new Popup Window has been shipped to Web Dynpro Java users. Compared to the existing Web Dynpro Popup Window, the new popup appears in a slightly different look-and-feel with additional functionalities. Dynamic programming is applied to add buttons below the “waveline”, bind properties to the window’s contexts, and link the user-defined action handlers to the window’s events.

**Author:** Dr. Wei-Guo Peng

**Company:** SAP AG

**Created on:** 20 November 2008

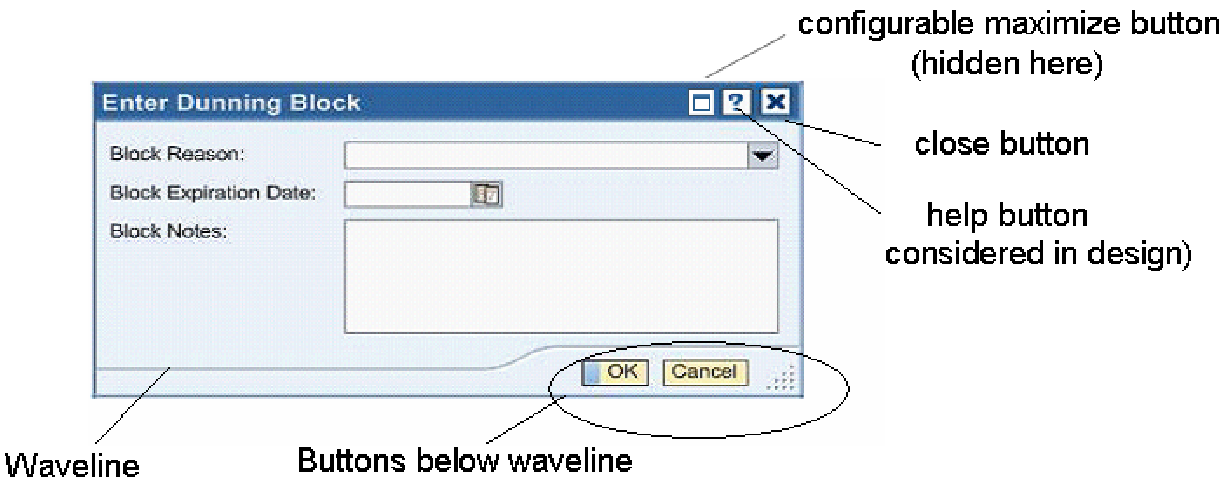
## Author Bio

Dr. Wei-Guo Peng joined SAP in 1998. He started working first in a joint project for digital printing between SAP and Heidelberg Printing Machine. Thereafter he has been involved in different development projects such as SAP Knowledge Warehouse, Mobile and Visual Composer for Web Dynpro. Since 2007 he is a member of the development team of Web Dynpro Java Runtime.

## Table of Contents

## Introduction

Compared to the existing Web Dynpro popup window, the new popup appears in a slightly different look-and-feel with additional functionalities. New APIs and user coding hooks Window Controller are introduced to enable the configuration of Window properties and actions. Currently no explicit support from NWDS (NetWeaver Developer Studio) is available. Most of the work should be done by dynamic programming.



Let’s have a look at the new features in detail below:

- A popup can have its own buttons which are rendered below the so called “waveline”. The buttons belong to the Window itself, not the View within the Window as it is the case with the “old” popup. Nevertheless the old popups still work.
- A popup can have a “Close Icon” in the top right corner to close the window. This was missing with the “old” popup.
- A popup can be configured to be one of the predefined “window types”: NONE, INFORMATION; WARNING; ERROR; SUCCESS. Web Dynpro runtime renders the popup with corresponding icon located on the left part of the window
- All the properties of the popup can be bound to context attributes. This was not possible with the “old” popups.
  The bindable properties include
  - title
  - resizable: if true you can resize the window by dragging the lower right corner
  - windowType: optional icon can be shown
  - hasContentPadding: if true a system defined padding will be applied, otherwise user has to care
  - width, height: size of the popup
  - left, right: position of the popup
  - maximized: if the popup should be opened in the maximized state
  - hasMinimumSize: if the minimum height & width should be applied. This minimum size is maintained by UR.
  - defaultButtonId: if specified the button is treated as default button. After open the popup this default button will be triggered by pressing <RETURN> key
- An action can be created “on the fly” if an event handler is defined with the IDE. The created actions can be used for the buttons below the waveline as well as for the close icon.
- Hooks wdDoModifyView() and wdDoBeforeAction() are now available in Window controller

## How to create a popup with the new feature mentioned above?

Before starting you should check the version of your NWDS. This can be done in NWDS: “Help” -> “About SAP NetWeaver Developer Studio”. The version information should look similar to the text below:

```
SAP NetWeaver Developer Studio
SAP NetWeaver 7.1 EhP 1 SP00 PAT0000
```

Make sure you use a NWDS with at least a version of 7.1 EhP 1. This is important since the older NWDS does not generate the necessary user hooks such as wdDoModifyView() etc.

For each popup window you have to create a Web Dynpro Window and “model” your window in the wdDoModifyView() hook of that Window controller by switching to the Java view.

Within the wdDoModifyView() hook, the first step is to fetch the IWDWindowViewElement from the View coming into the hook as the first parameter, where the casting to IWDWindowViewElement is necessary. This UI element is responsible for everything related to (popup) window including all the bindable properties, buttons below the waveline, as well as the close action of the window

```
public void wdDoModifyView(com.sap.tc.webdynpro.progmodel.api.IWDView view, boolean
firstTime)
{
  //@@begin wdDoModifyView
if (firstTime) {
  // IWDWindowViewElement is the root UIElement of the View
  IWDWindowViewElement window = (IWDWindowViewElement)
    view.getRootElement();

  // create an action from the event handler
  IWDAction okAction = wdThis.wdGetAPI().createAction("OK Action",
     IPrivatePopupWindow.WD_EVENTHANDLER_ON_OK, true); // validating
  IWDAction cancelAction = wdThis.wdGetAPI().createAction("Cancel Action",
    IPrivatePopupWindow.WD_EVENTHANDLER_ON_CANCEL, false); // non-validating

  // insert OK button below the wave line
  IWDButton okButton = view.createElement(IWDButton.class, "okButtonId");
  okButton.setOnAction(okAction);
  okButton.setText("OK");
  window.addButton(okButton);
  window.setDefaultButtonId("okButtonId");
  view.requestFocus(okAction);

  // insert CANCEL button below the wave line
  IWDButton cancelButton = view.createElement(IWDButton.class, null);
  okButton.setText("Cancel");
  cancelButton.setOnAction(cancelAction);
  window.addButton(cancelButton);

  // system close event
  window.setOnClose(cancelAction); //links to action

  // bindable window properties
  IWDAttributeInfo attrInfo;

  // bind the context attributes “title” to window property title
  attrInfo = wdContext.getContext().getRootNodeInfo().getAttribute("title");
  window.bindTitle(attrInfo);

  // bind the context attributes “maximized” to window property maximized
  attrInfo = wdContext.getContext().getRootNodeInfo().getAttribute(“maximized");
  window.bindMaximized(attrInfo);
  // alternatively the window properties can be also set
  window.setHasContentPadding(true);
  window.setWindowType(WDWindowType.SUCCESS);

  // system close icon 'x' can be deactivated
  cancelAction.setEnabled(false);

  //@@end
}
```

All the mentioned properties can be bound conveniently to the local window context just like any UI element in the View controller. Web Dynpro framework takes care of the data transfer between WindowViewElement and window's context. Alternatively each property can be also set to a value without data synchronization with the context.

The buttons below waveline must be created dynamically (no layout modelling possibility yet in NWDS, see section limitations for detail). The method addButton() of IWDWindowViewElement places a button below the popup's waveline. Any numbers of buttons are allowed, but beware of the limited space and usability guidelines. IWDWindowViewElement.setDefaultButtonId() is used to mark a button as default button.

The buttons of (popup) window need usually actions. An action can be created on the fly with IWDWindowController.createAction() method. This method requires the existence of an event handler. The last parameter indicates if the action is a validating or non-validating action.

The "Close Icon" will be shown if the following steps are done:

1. IWDWindowViewElement.setOnClose() is called with a valid close action as parameter
2. The close action is enabled.

The "Close Icon" can be turned on or off dynamically by changing the states of above mentioned properties. For instance, to set the close action to be disabled or call setOnClose(null).

## How to deal with action and event handler?

It might be a little bit confusing for someone, who is familiar with modelling UI layout in View controller, that there is no way to define Action in Window controller with NWDS. It is a common practice with View controller in NWDS that an action is created, which can be then assigned to an event of a UIElement (button onAction). NWDS generates the corresponding event handler as well as a user hook, where application Java code can be inserted. The event handler Id is generated as a static variable and can be referred either by wdThis.WD_EVENTHANDLER_<NAME> or IPrivatePopupWindow. WD_EVENTHANDLER_<NAME>. To assign actions for the popup window buttons (Ok, Cancel, Close etc) actions must be created dynamically.

- Declare an event handler in Window controller: In NWDS switch to Window controller view, click the "Methods" tab, click "New…" and check "Event handler" type. Switch to the Java view you can see a user hook also generated.
- Create an action dynamically based on the event handler defined in the previous step and associate it with the popup button: This must be done with Java code and in the wdDoModifyView() hook of the Window controller.

```
...
IWDAction okAction = wdThis.wdGetAPI().createAction("OK Action",
    wdThis.WD_EVENTHANDLER_ON_OK, true);
okButton.setOnAction(okAction);
...
```

## Open a popup window

To open (show) or close (hide) a popup window remains unchanged. E.g. in the action handler to open the popup you must find the window definition IWDWindowInfo from the runtime repository and pass it as a parameter to the method createModalWindow() of IWDWindowManager. The IWDWindow object is the Window instance returned by opening the popup. To be able to hide the opened popup you have to store the instance in a context of the Web Dynpro Component for later use.

To open (show) or close (hide) a popup window remains unchanged. E.g. in the action handler to open the popup you must find the window definition IWDWindowInfo from the runtime repository and pass it as a parameter to the method createModalWindow() of IWDWindowManager. The IWDWindow object is the Window instance returned by opening the popup. To be able to hide the opened popup you have to store the instance in a context of the Web Dynpro Component for later use.

```
//@@begin javadoc:onActionOpenPopup(ServerEvent)
/**
 * Declared validating event handler.
 *
 * @param wdEvent generic event object provided by framework
 */
//@@end
public void onActionOpenPopup(com.sap.tc.webdynpro.progmodel.api.IWDCustomEvent
wdEvent, java.lang.String popupName )
{
  //@@begin onActionOpenPopup(ServerEvent)
  IWDWindowInfo windowInfo =
  (IWDWindowInfo)wdComponentAPI.getComponentInfo().findInWindows(popupName);

  IWDWindow window = wdComponentAPI.getWindowManager()
    createModalWindow(windowInfo);
  window.show();
  wdContext.currentContextElement().setPopupInstance(window);
  //@@end
}
```

The statement window.show() opens the popup window. The last statement stores the popup window in the context, so that the popup window can be closed later on.

## New APIs

The following API interfaces are new:

- IWDWindowViewElement
- WDWindowType

Refer to the JavaDoc for detailed information.

In IWDWindowController a new method can be used to create Action from an EventHandler ID:

```
/**
 * Creates a new action with name <code>name</code> and event handler
ID<code>eventHandlerId</code>.
 * @param name name of the action, eventHandlerId id of the event handler for this
action
 * @return newly created action
 */
IWDAction createAction(String actionName, IWDEventHandlerId eventHandlerId, boolean
isValidating);
```

The following methods in IWDWindow API shouldn't be used to manipulate the window properties:

- setWindowPosition()
- setTitle()
- setWindowSize()

Instead use IWDWindowViewElement API

- setTitle, bindTitle()
- setLeft(), bindLeft(); setTop(), bindTop()
- setWidth(), bindWidth(); setHeight(), bindHeight()

It is strongly recommended NOT to mix up the use of old APIs with the new ones.

## Confirmation Dialogs

Web Dynpro Confirmation Dialog is a predefined special Popup Window, which contains only a short text and several buttons. It is a short cut to the normal popup. The main advantage using confirmation dialog is that there is no need to define a Web Dynpro Window, instead user creates this Window by calling a method of the API:

```
//@@begin javadoc:onActionOpenConfirmation(ServerEvent)
/**
 * Declared validating event handler.
 *
 * @param wdEvent generic event object provided by framework
 */
//@@end
public void onActionOpenConfirmation(
  com.sap.tc.webdynpro.progmodel.api.IWDCustomEvent wdEvent ) {
  //@@begin onActionOpenConfirmation(ServerEvent)

  IWDConfirmationDialog window = wdComponentAPI.getWindowManager()
    .createConfirmationWindow(infoText.toString(),
     wdThis.wdGetCloseAction(), "Close");
  window.addChoice(wdThis.wdGetCancelAction());

  if (wdContext.currentContextElement().getHasCloseIcon()) {
    window.setOnClose(wdThis.wdGetCloseAction());
    IWDAction action = wdControllerAPI.getAction("Close");
    window.setOnClose(action);
  }

    window.setWindowSize(500, 200);
    window.show();
  //@@end
}
```

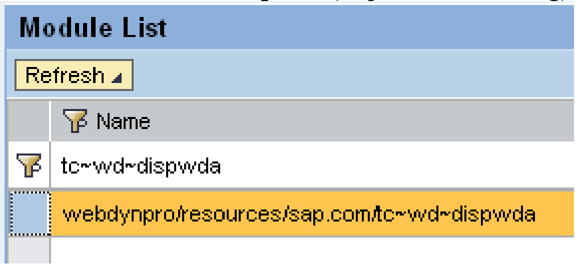
There exist two variants of the same API to create a confirmation cialog. One needs IWDAction as parameter like code above. This is called the action-based confirmation dialog. The other option is to pass the event handler ID as parameter:

```
//@@begin javadoc:onActionOpenConfirmation(ServerEvent)
/**
 * Declared validating event handler.
 *
 * @param wdEvent generic event object provided by framework
 */
//@@end
public void onActionOpenConfirmation(
  com.sap.tc.webdynpro.progmodel.api.IWDCustomEvent wdEvent ) {
  //@@begin onActionOpenConfirmation(ServerEvent)
......
  window.addChoice(wdThis.WD_EVENTHANDLER_ON_CANCEL, "Cancel");
  if (wdContext.currentContextElement().getHasCloseIcon()) {
    window.setOnClose(wdThis.WD_EVENTHANDLER_ON_CANCEL, true);
  }
......
  //@@end
}
```

It is not allowed to mix them up for the same confirmation dialog.

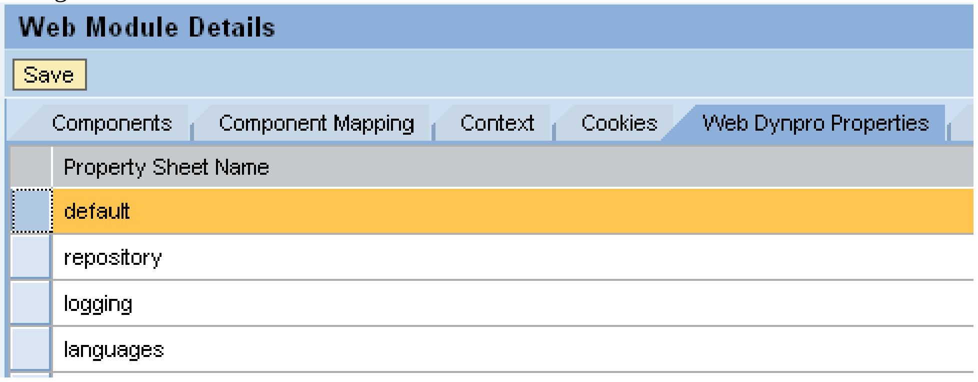
The new design with waveline should be the default case, if not it can be switched on by setting the global parameter defined in the default.properties: sap.corecomp.useWavelineDesign = true. This parameter can be configured using the *NetWeaver Administrator (NWA)*:

- Log in to *NetWeaver Administrator* either by choosing *NetWeaver Administrator* on your engine's *Start page* or by entering the following URL: http://<your host name>:<port>/**nwa**
- Select *Configuration Management* in the top level navigation and *Infrastructure* in the second level navigation bar.
- Select *Application Modules* and enter **tc~wd~dispwda** (or parts of this string) into the filter row.



| | Name |
|---|---|
| | tc~wd~dispwda |
| | webdynpro/resources/sap.com/tc~wd~dispwda |

- In the section *Web Module Details* select the *Web Dynpro Properties* Tab and select *default* from the corresponding table.



| Property Sheet Name |
|---|
| default |
| repository |
| logging |
| languages |

- Navigate in the Full Details section to the property sap.corecomp.useWavelineDesign and set it to true
- To confirm your changes, click *Save* in the *Web Module Details* section
- Restart the Engine (or only the node of the server)

## Known limitations

No NWDS support is currently available. The View Designer will be enhanced in the later releases. Most of the development effort must be done through dynamic programming.

UR Classic rendering (instead of the default Lightspeed) is supported only with limitations:

- Close button/icon is not supported. Therefore it is strongly recommended that all the popup windows should contain at least a close button below the waveline.
- Most properties are not interactive and require reopen of the popup.

All properties defined in IWDWindowViewElement are not personalizable.

The properties of left and top effect window position only if they are set before window is opened.

The properties of width and height can have effect on the current window if they are larger or the window is reopened.

If it is embedded, the buttons of embedded will be ignored. If it is the Application Window, an exception will be thrown if buttons exist.

Mixing up of these two APIs to set or bind the window properties could result in inconsistency.

## Related Content

You can find more new features of Web Dynpro for Java in SAP NetWeaver Composition Environment 7.1 in the article:

What's New in Web Dynpro Java?

For more information, visit the User Interface Technology homepage

## Copyright