# Dialog Windows in Web Dynpro Java Applications

## An Explanation and How To Guide on how to create popup dialog windows, external windows and confirmation boxes.

Releases: SAP Netweaver Composition Environment 7.1
SAP NetWeaver 7.1

# Copyright

## Icons in Body Text

| Icon | Meaning |
| --- | --- |
|  | Caution |
|  | Example |
|  | Note |
|  | Recommendation |
|  | Syntax |

Additional icons are used in SAP Library documentation to help you identify different types of information at a glance. For more information, see *Help on Help* → *General Information Classes and Information Classes for Business Information Warehouse* on the first page of any version of *SAP Library*.

## Typographic Conventions

| Type Style | Description |
| --- | --- |
| *Example text* | Words or characters quoted from the screen. These include field names, screen titles, pushbuttons labels, menu names, menu paths, and menu options.<br>Cross-references to other documentation. |
| **Example text** | Emphasized words or phrases in body text, graphic titles, and table titles. |
| EXAMPLE TEXT | Technical names of system objects. These include report names, program names, transaction codes, table names, and key concepts of a programming language when they are surrounded by body text, for example, SELECT and INCLUDE. |
| `Example text` | Output on the screen. This includes file and directory names and their paths, messages, names of variables and parameters, source text, and names of installation, upgrade and database tools. |
| **`Example text`** | Exact user entry. These are words or characters that you enter in the system exactly as they appear in the documentation. |
| **`<Example text>`** | Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system. |
| `EXAMPLE TEXT` | Keys on the keyboard, for example, `F2` or `ENTER`. |

# Dialog Boxes in Web Dynpro Applications

In this tutorial, you use a few steps to create a Web Dynpro application that uses various types of dialog box.

## The Task

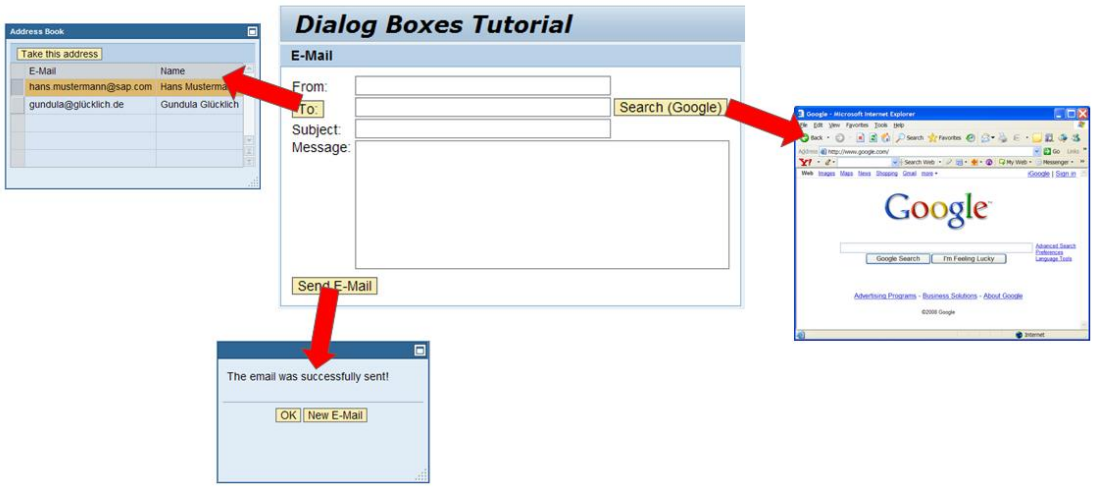
In the tutorial Development Component project **wd/tut/popup_init**, the starting point is the application for sending an e-mail. This application contains an input form for the e-mail addresses of the sender and the recipient, the subject, and the text.

The task of this tutorial is to assign functions to the *To, Search (Google)*, and *Send Email* pushbuttons (see figure).

If you choose *to*, a new Web Dynpro window containing a simple address book is displayed in a dialog box. If you select an address in this dialog box, this is written to the input field for the recipient address.

If you choose *Search (Google)*, an external window, which is assigned a URL address (for example, http://www.google.de), opens.

If you choose *Send Email*, a confirmation dialog box containing two pushbuttons appears. *Ok* closes the dialog box and *new email* closes the dialog box and deletes the content of the input form.




## Objectives

When you have completed the described procedures, you will be able to:

- ✓ Display a Web Dynpro window in a dialog box
- ✓ Open an external window in a Web Dynpro application
- ✓ Create a confirmation dialog box

## Prerequisites

### Systems, Installations, and Authorizations

- SAP NetWeaver Developer Studio 7.1 or greater is installed on your PC.

- You have access to the SAP AS Java 7.1.

### Knowledge

- Basic knowledge of the Java programming language
- Knowledge of programming Web Dynpro applications

### Next step:

Importing a Project Template [Page 6]

# Importing a Development Component Project Template

The exercise and solution projects are both available for download from SDN. See below for download links.

- The Development Component project template *wd/tut/popup_init* (the starting point for this tutorial) – Click here to download
- The completed Web Dynpro project *wd/tut/popup* (corresponds to the project *wd/tut/popup_init* after completion of the tutorial) – Click here to download

## Prerequisites

- You have a user ID and password to access the SAP Developer Network (http://sdn.sap.com).
- You have installed the SAP NetWeaver Developer Studio.

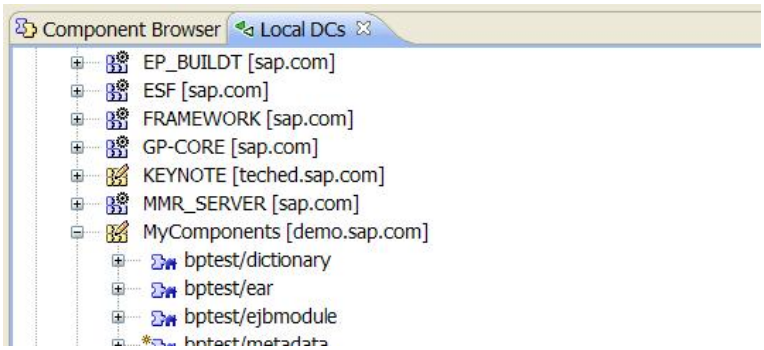
## Procedure

### Importing the Project Template into the SAP NetWeaver Developer Studio

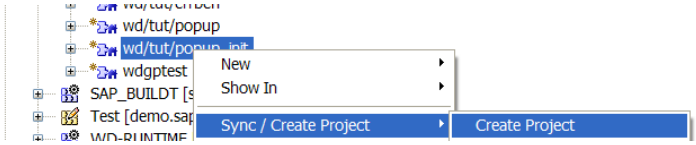
1. Download the project zip file from the locations links listed above. Save the ZIP file containing the Development Component project to your file system.
2. Extract the content of the zip file into the Local Development work area of the SAP NetWeaver Developer Studio. The actual folder location depends on the Developer Studio configuration; that said here is an example of where the zip should be extracted to:

   **C:\Documents and Settings\<User Name>\workspaceCE.jdi**

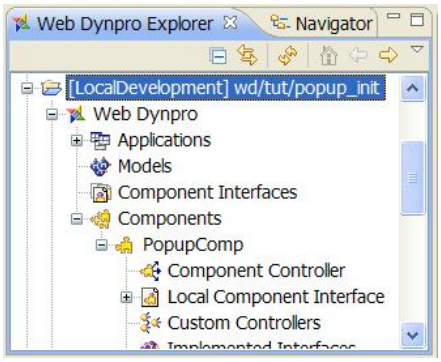
3. Start the SAP NetWeaver Developer Studio.
   - Open the Development Infrastructure perspective and select the Local DCs view.




- Under the *MyComponents* find the Development Component wd/tut/popup_init and in its context menu select *Sync/Create Project > Create Project*




- This will create the project for the Development Component in the Developer Studio.
- Once done, switch to the Web Dynpro perspective.




  - The Web Dynpro Development Component project *wd/tut/popup_init* appears in the Web Dynpro Explorer for further processing and completion of the tutorial.

## Tabular Project Structure

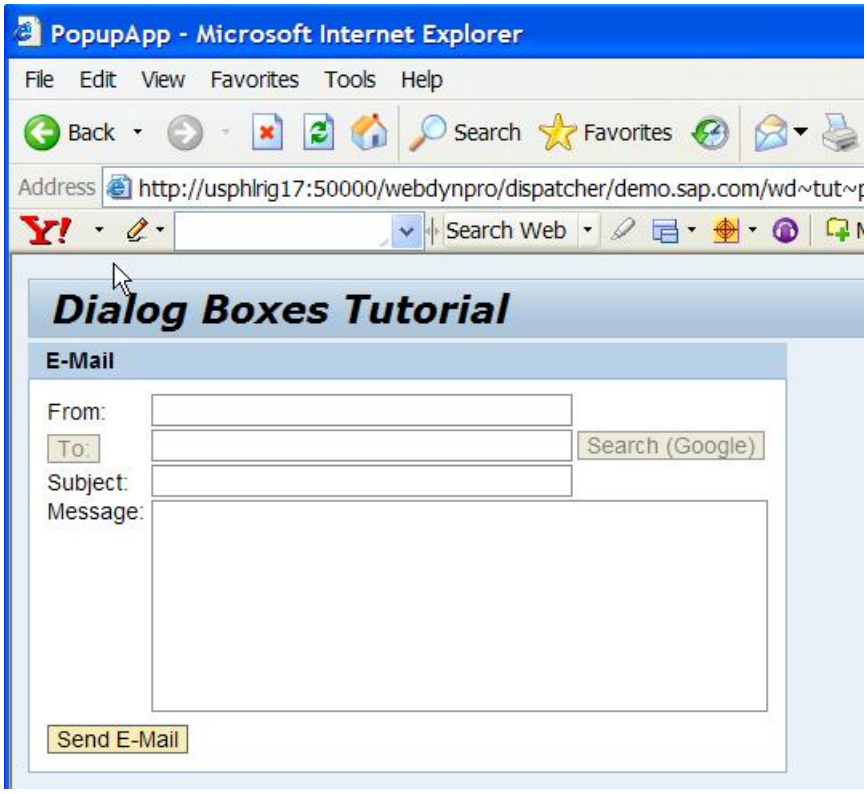
Once you have imported the project template into the SAP NetWeaver Developer Studio, you will see the following structure in the Web Dynpro Explorer.

| **Web Dynpro Project Structure** |
|---|
| **Web Dynpro Development Component: wd/tut/popup_init** |
| **Web Dynpro Application: PopupApp** |
| **Web Dynpro Component: PopupComp** |
| **View: EmailView**<br>**This view contains the input form for sending an e-mail.** |
| **Windows: EmailWindow**<br>**The *EmailView* is embedded in this Web Dynpro window.** |

# Overview of the Imported Project

## Application

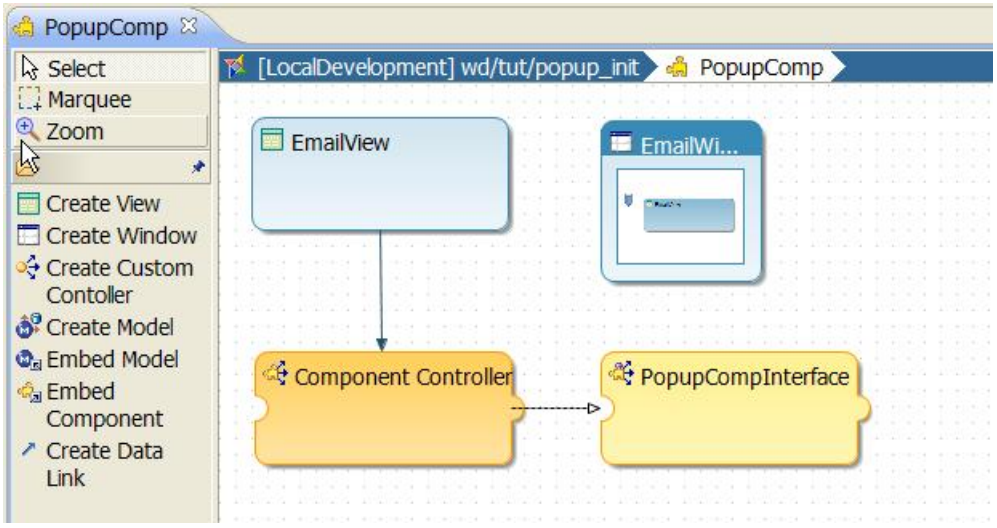
You can already execute the imported application *PopupApp*. The following window opens in the Internet browser:




The functions for sending the e-mail are not implemented in this tutorial. For more information about this topic, see the tutorial [Using an e-mail Web Service with Web Dynpro [Extern]].

## Data Modeler

The Data Modeler provides a graphical overview of the imported Development Component *wd/tut/popup_Init*.

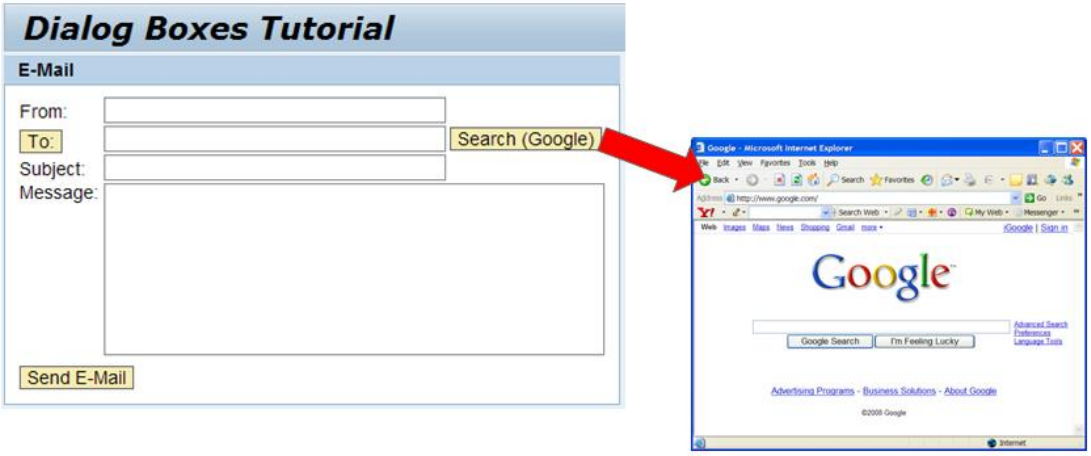


### Next Step:

Create an external window [Page 9]

# Creating an External Window

You want the web page http://www.google.com to be displayed in an external window when you choose *Search (Google)*.




## Procedure

You want the external window to be displayed when you choose *Search (Google)*. In the procedure below, you assign the action *ShowGoogleWindow* to the pushbutton *Search (Google)*.

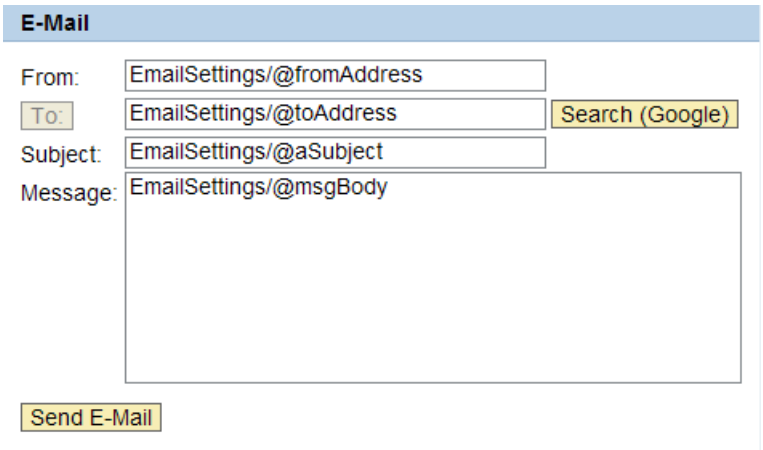
### Creating an Action

1. Define the action **`ShowGoogleWindow`** in the *EmailView* controller:

| Name of the Action | Validation | Event Handler |
|---|---|---|
| **`ShowGoogleWindow`** | *checked* | *onActionShowGoogleWindow* |

### Binding the Action to the Pushbutton

The e-mail input form is displayed in the *EmailView* of the Web Dynpro component *PopupComp*.

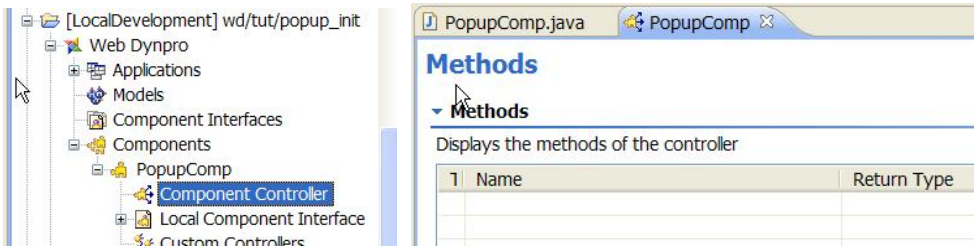


2. Bind the action that you defined earlier to the pushbutton *Search_Button*:

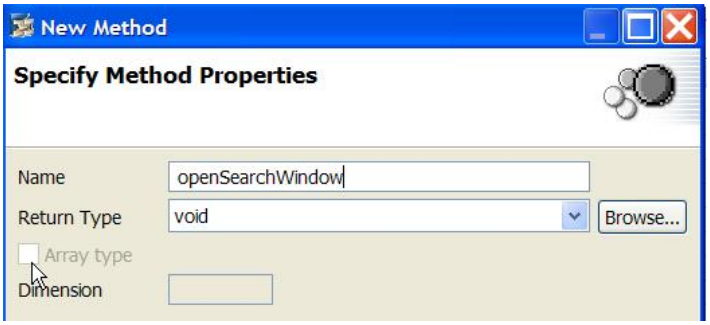
| UI Element Name | Event Name | Action |
|---|---|---|
| **`Search_Button`** | *onAction* | *ShowGoogleWindow* |

## Creating and Implementing the Open Search Window method on the Component Controller

3. Open the Component Controller by double clicking on the icon in the Web Dynpro Development Component structure.




4. Switch to the Methods tab of the Component Controller editor and create a new method named *openSearchWindow*.




The method should return *void* and have no parameters.

5. Switch to the implementation class of the Component Controller: *PopupComp.java*

6. Implement the `openSearchWindow()` method.

**`PopupComp.java`**

```
...
   public void openSearchWindow() {
      // @@begin openSearchWindow()
      IWDWindow window = wdComponentAPI.getWindowManager()
            .createNonModalExternalWindow(
                  "http://www.google.com",
                  "Google - Search for an email address");
      window.show();
      // @@end
   }
...
```

If the `openSearchWindow()` method does not exist in the implementation, rebuild the project (project context menu > Rebuild Project), and then open the implementation again.

You use the method **createNonModalExternalWindow**(java.lang.String URL, java.lang.String title) to create the external window from `IWDWindowManager`.

If needed you can add another method which closes and destroys the window & `IWDWindow` instance.

### Implementing the Action Handler

7. Switch to the implementation class of the view EmailView: *EmailView.java*
8. In the method *onActionShowGoogleWindow(…)*, add code to call the Component Controller's *openSearchWindow()* method.

**EmailView.java**

```
...
public void onActionShowGoogleWindow(
         com.sap.tc.webdynpro.progmodel.api.IWDCustomEvent
            wdEvent )
  {
    //@@begin onActionShowGoogleWindow(ServerEvent)
     wdThis.wdGetPopupCompController().openSearchWindow();
    //@@end
  }
...
```

## Result

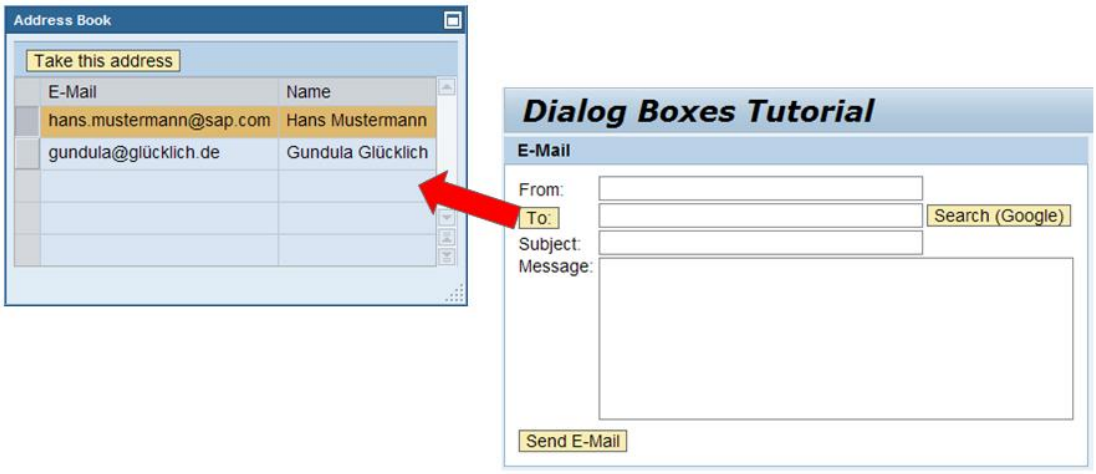
You have enhanced the existing Web Dynpro application to display an external window. You should be able to build, deploy and test the application.

### Next step:

Creating a Dialog Box [Page 11]

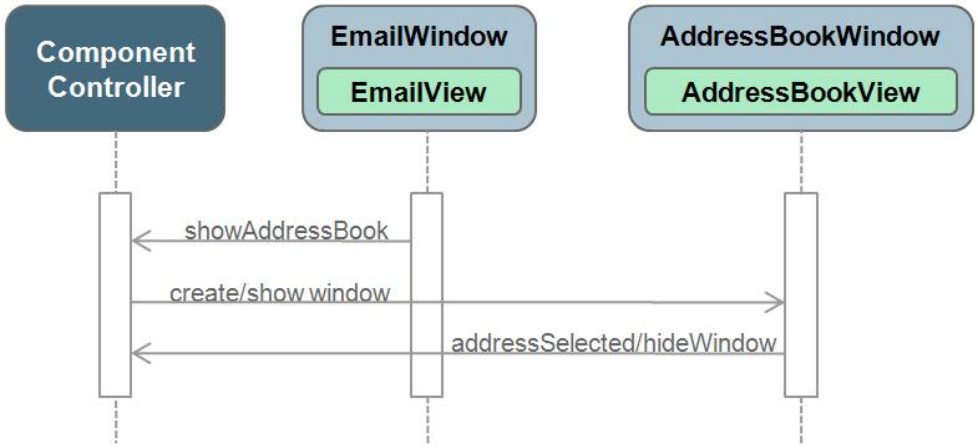
# Creating a Dialog Box

You want an address book to appear in a dialog box when you choose *To*. In this address book, you can select an e-mail address. When you choose *Take this address*, the dialog box closes and the e-mail address is written in the recipient input field.




The address book is embedded in a separate Web Dynpro window, which is created and opened when you choose the *To* pushbutton. To inform the *EmailView* that an address has

been selected, the component controller is used to trigger an event to *EmailView*, which then closes the *AddressbookWindow*.




If you no longer need the dialog box, you close it using `destroyInstance()`. This allows you to reuse the *InterfaceView*: for example, in another dialog box or in a view in which this *InterfaceView*is embedded.

If you use `hide()`, the dialog box is hidden and the *InterfaceView* is no longer visible. You can reopen the dialog box using `show()`. However, you cannot embed the *InterfaceView* in another view or in another dialog box.

**The procedure for creating this dialog box is divided into the following steps:**

- Create and implement the Web Dynpro window for the address book
  - Create the Web Dynpro window
  - Create and embed the Web Dynpro view
  - Create the context for the address book
  - Design the layout of the *AddressbookView*
  - Implement the controller of the *AddressbookView*
- Define the interaction of the two Web Dynpro windows *EmailWindow* and *AddressbookWindow*
  - Enhance the context of the *Component Controller*
  - Creating and Implementing *showAddressbookPopup()* method on the Component Controller
  - Creating and Implementing *hideAddressbookPopup()* method on the Component Controller
  - Creating and implementing the action and event handler *ShowAddressbookPopup* on the EmailView View Controller
  - Create and implement the action and event handler *AddressSelected* on the AddressbookView View Controller

## Next step:

Creating and Implementing a Web Dynpro Window for the Address Book [Page 13]

# Creating a Web Dynpro Window for the Address Book

## Use

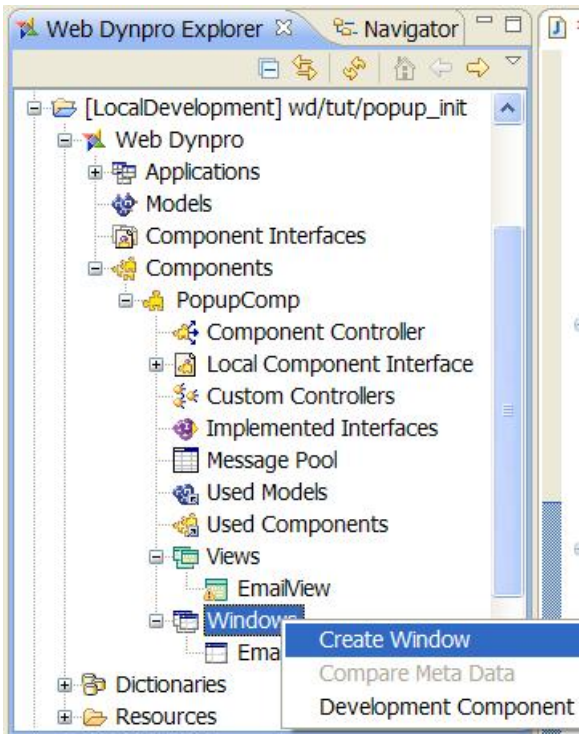
To display an address book in a Web Dynpro window, you must define this in a view. To create the address book, you use the following procedure:

- Create the Web Dynpro window
- Create and embed the Web Dynpro view
- Creating Context for the Address Book
- Designing the Layout of the AddressbookView
- Implementing the Controller of the AddressbookView

## Procedure

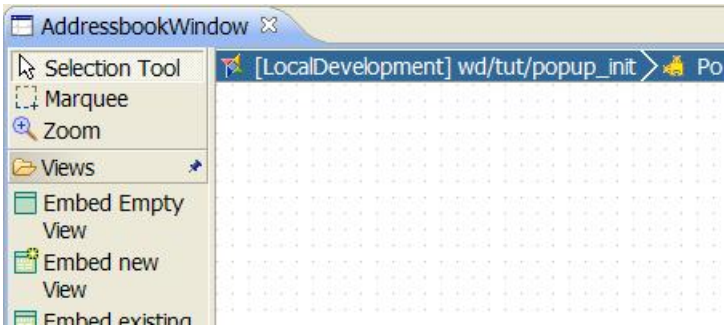
### Creating a Web Dynpro Window

1. In the Web Dynpro Explorer, expand the following nodes: Web Dynpro → Web Dynpro Components → PopupComp → Windows.
2. Open the context menu of the *Windows* node and choose *Create Window.*
3. In the Create wizard, give the window the name **`AddressbookWindow`**.
4. Leave the default settings unchanged and choose *Finish* to confirm.
5. Change the *Title* of the *AddressbookWindow* in the *Properties* view by changing *title* to **`Address Book`**.

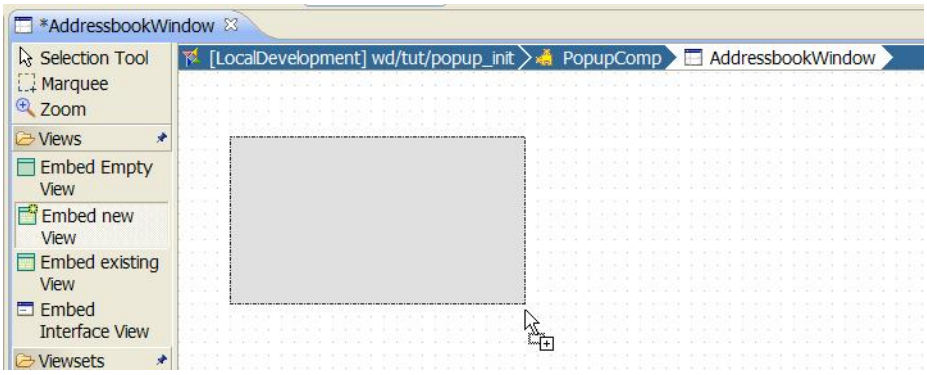



### Creating and Embedding a Web Dynpro View

1. Open the Web Dynpro window **`AddressbookWindow`** and go to the Navigation tab of the Window editor.




2. Select Embed new view and using the mouse click and drag on the Window diagram to create a new view.

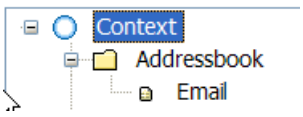



3. Enter the name **`AddressbookView`**, leave the default settings unchanged, and choose *Finish* to confirm.

## Creating the Context for the Address Book

You want the address book to display names and e-mail addresses. You want the selected e-mail address to be displayed in the *AddressbookView* and in the *EmailView*. Thus, the context for the email address is created in the component controller.

1. Open the *Component Controller* of the *PopupComp* (*Web Dynpro Components* → *PopupComp* → *Component Controller*).
2. Choose the *Context* tab page and create the context node **`Addressbook`** and the context attribute **`Email`**:

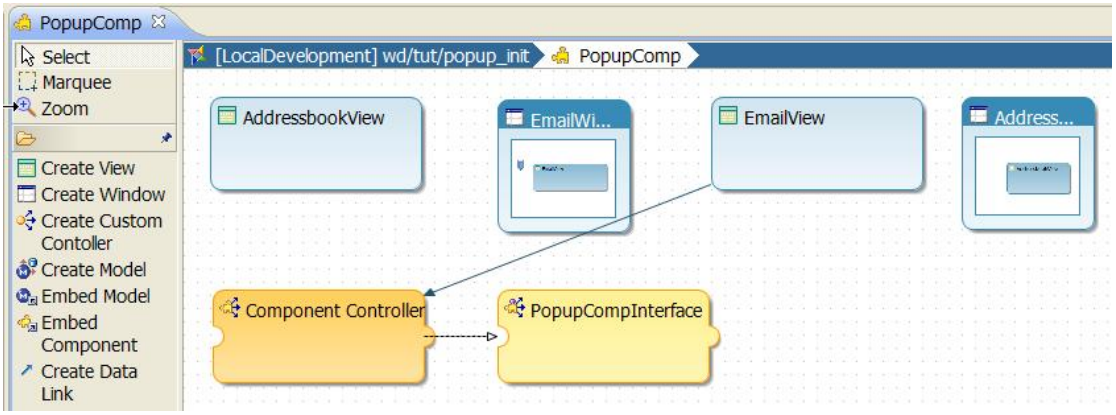



| Context Element | Type | Properties | Value |
|---|---|---|---|
| Addressbook | Value node | *cardinality* | `0..n` |
| E-mail | Value attribute | *type* | `String` |

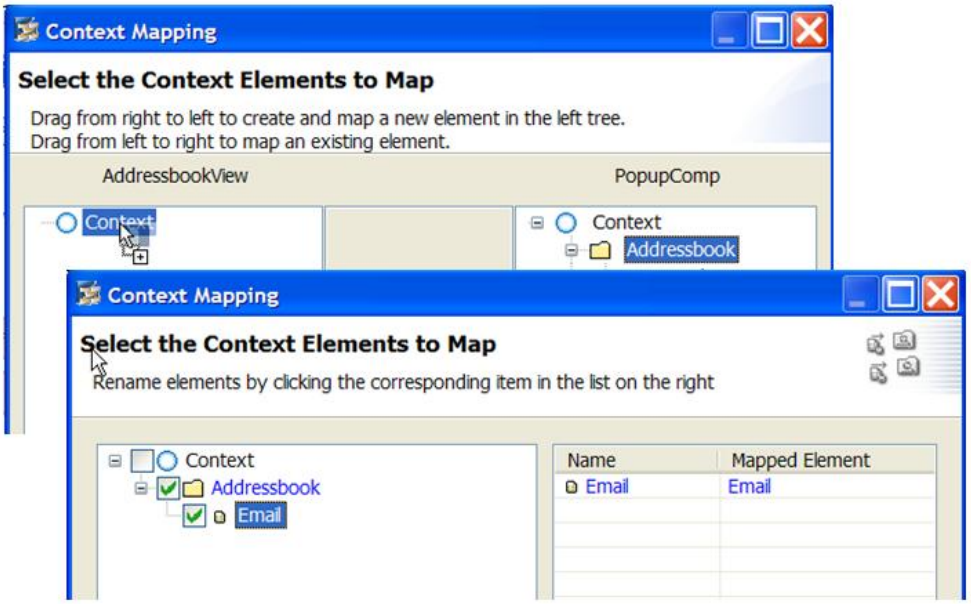
Once you have enhanced the context in the *Component Controller*, you can map the context of the *AddressbookView* to it :

3. To open the **Data Modeler** for the *PopupComp*, in the context menu of the **PopupComp**, choose *Open Data Modeler*.

   The *PopupComp* overview opens in the Data Modeler:

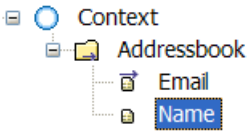



4. Create a *Data Link* from the *AddressbookView* to the *Component Controller*. Drag the context node **Addressbook** from the *Component Controller* to the *AddressbookView* and drop it there*:* In the following dialog box, select the context attribute **Email** and the context node **Addressbook**.




Since you only want to display the name belonging to the e-mail in the *AddressbookView*, you only define it in the context of the *AddressbookView*.

5. Open the *Context Editor* of the *AddressbookView* and add the context attribute **`Name`**, which belongs to the *Addressbook* context node, to the context.



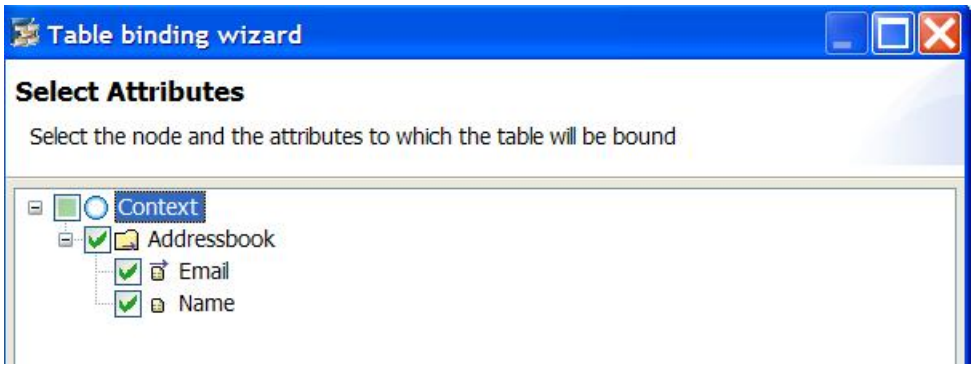

| Context Element | Type | Properties | Value |
|---|---|---|---|
| Name | Value attribute | *type* | `String` |

## Designing the Layout of the *AddressbookView*

The layout of the *AddressbookView* consists of a table with two columns. The first column contains the e-mail address and the second column contains the corresponding name. The

table also has a toolbar pushbutton *Take this address*, which you use to confirm the e-mail address selection.


1. Switch to the *Layout* tab page of the *AddressbookView*.
2. Delete the default text view element.
3. Create a table called **`Table`**.
4. In the context menu of the table **Table**, choose *Create Binding*.
5. In the Table binding wizard, choose the Addressbook context node with the value attributes *Name* and *Email*. Choose *Finish* to confirm.



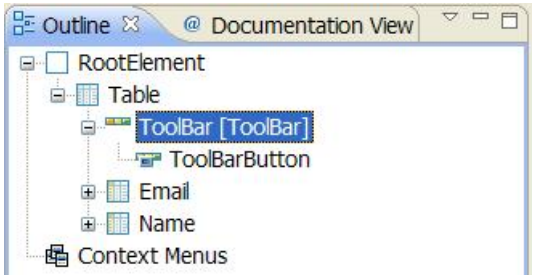

This creates the *Email* and *Name* table columns (see figure 6) automatically.

6. Now create a toolbar in which you want a pushbutton to be displayed. From the Outline view go to the context menu of the table **Table**, choose *Insert ToolBar*.

   This creates a toolbar for the table.

7. In the Toolbar context menu, choose *Insert ToolBarItem*. In the wizard that appears, create a *ToolBarButton* called **`ToolBarButton`**.
8. Change the properties of the *ToolBarButton*:

| Properties | Value |
|---|---|
| **ToolBarButton** of type *ToolBarButton* | |
| *Text* | Take this address |



### Implementing the Controller of the *AddressbookView*

1. To fill the address book with example addresses, add the following program code to the method `wdDoInit()` in `AddressbookView.java`.

**AddressbookView.java**

```
...
public void wdDoInit()
{
   //@@begin wdDoInit()
   wdContext.nodeAddressbook().invalidate();

   IPrivateAddressbookView.IAddressbookNode
     addressNode = wdContext.nodeAddressbook();
   IPrivateAddressbookView.IAddressbookElement addressElement;

   //Person 1
   addressElement = addressNode.createAddressbookElement();
   addressNode.addElement(addressElement);
   addressElement.setName("Hans Mustermann");
   addressElement.setEmail("hans.mustermann@sap.com");

   //Person 2
   addressElement = addressNode.createAddressbookElement();
   addressNode.addElement(addressElement);
   addressElement.setName("Gundula Glücklich");
   addressElement.setEmail("gundula@glücklich.de");

   //@@end
}
...
```

This writes the example addresses to the context when the *AddressbookView* is initialized.

At the beginning, the context must be invalidated, as otherwise the mapped context of the component controller would be displayed as well when repeatedly opening the dialog box.

## Result

You have created a Web Dynpro window that embeds the *AddressbookView*, which displays an address book.

### Next Step:

Interaction of the EmailWindow and the AddressbookWindow [Page 18]

# Interaction of the EmailWindow and the AddressbookWindow

To open and close the address book (*AddressbookWindow*) and copy the e-mail address from the address book to the *EmailView* (embedded in the *EmailWindow*), you must make the following enhancements:

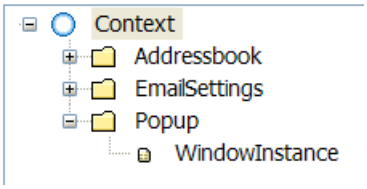
- Enhance the context of the *Component Controller*
- Creating and Implementing *showAddressbookPopup* method on the Component Controller
- Creating and Implementing *hideAddressbookPopup* method on the Component Controller
- Creating and implementing the action and event handler *ShowAddressbookPopup* on the EmailView View Controller
- Creating and implementing the action and event handler *AddressSelected* on the AddressbookView View Controller

## Procedure

### Enhancing the Context of the *Component Controller*

To be able to create the address book window and close it again, you must save the window instance in the context of the *Component Controller*.

1. To do this, enhance the context in the *Component Controller*:




| Context Element | Type | Properties | Value |
|---|---|---|---|
| Popup | Value node | *cardinality* | `1..1` |
| WindowInstance | Value attribute | *type* | `com.sap.tc.webdynpro.services.session.api.IWDWindow` |

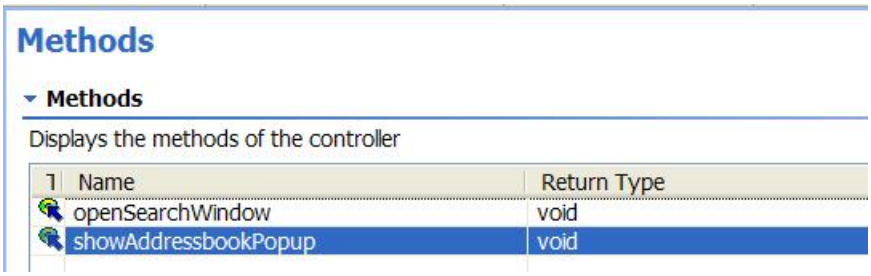
#### Assigning a *Java Native Type* as the Data Type to the attribute *WindowInstance*

The values of the value attribute *WindowInstance* require a Java Native Type as the data type. Proceed as follows:

1. In the Context Attribute creation dialog, select the Manually radio button then click the Browse… button to select a type (Java Native Class in this case).
2. Select the *Java Native Type* radio button and choose *Browse…* This opens another window.
3. In the input field, enter **`IWDWindow`** for the value attribute. Choose *OK* to confirm.
4. Choose *Finish* to confirm.

### Creating and Implementing the *showAddressbookPopup()* method.

1. Open the Component Controller editor and go to the Methods tab.
2. Create the new method called *showAddressbookPopup()*, it should have no parameters and a void return type.




3. Open the implementation class file for the Component Controller: *PopupComp.java*
4. Implement the *showAddressbookPopup()* method. Add code to create and show the popup window.

**PopupComp.java**

```
...
public void showAddressbookPopup() {
   // @@begin showAddressbookPopup()
   // get the repository content at runtime of the Web-Dynpro-
   // Window “AddressbookWindow”
   IWDWindowInfo windowInfo = (IWDWindowInfo) wdComponentAPI
         .getComponentInfo().findInWindows("AddressbookWindow");
   // create the “AddressbookWindow”
   IWDWindow window = wdComponentAPI.getWindowManager()
      .createModalWindow(windowInfo);
   // set the WindowPosition on the screen
   window.setWindowPosition(300, 150);
   // and show the window
   window.show();

   // Save WindowInstance in Context
   wdContext.currentPopupElement().setWindowInstance(window);
   // @@end
}
...
```

You use the method `findInWindows()` to obtain the repository content of the Web Dynpro window *AddressbookWindow*.

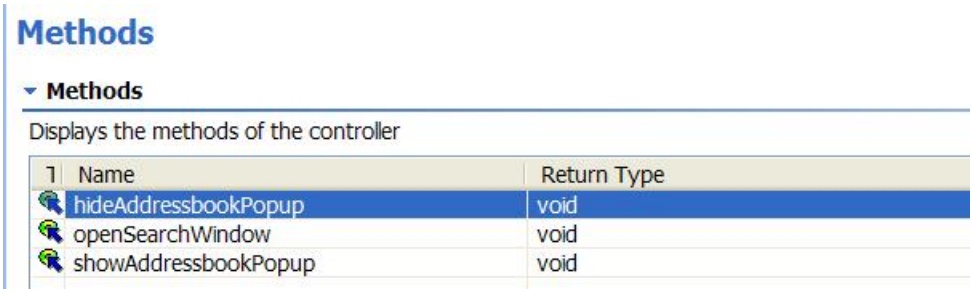
You use the method `createModalWindow()` of the *IWDWindowManager* to create the Web Dynpro window and `show()` to display it.

Having `createModalWindow()` twice in the same window instance would cause an error if the first instance is not opened with `show()`. In one window instance, a `createModalWindow()` must always be followed by a corresponding `destroyInstance()`.

To close the window again, you need the window instance. Therefore, this is saved in the context.

## Creating and Implementing the *hideAddressbookPopup()* method.

1. Open the Component Controller editor and go to the Methods tab.
2. Create the new method called *hideAddressbookPopup()*, it should have no parameters and a void return type.




3. Open the implementation class file for the Component Controller: *PopupComp.java*
4. Implement the *hideAddressbookPopup()* method. Add code to store the email address that was selected in the Addressbook popup window and then close and destroy the window (this can be done with one call the the Window's *destroyInstance()* method).

**PopupComp.java**

```
...
public void hideAddressbookPopup() {
   // @@begin showAddressbookPopup()
   // put the selected email into context-attribute "ToAddress"
   wdContext.currentEmailSettingsElement().setToAddress(
         wdContext.currentAddressbookElement().getEmail());

   IWDWindow window = wdContext.currentPopupElement()
         .getWindowInstance();
   window.destroyInstance();
   // @@end
}
...
```

## Creating and Implementing the action and event handler *ShowAddressbookPopup on the EmailView View Controller*

1. In the *EmailView*, create the action **ShowAddressbookPopup**:

| Name of the Action | Validation | Event Handler |
|---|---|---|
| ShowAddressbookPopup | checked | *onActionShowAddressbookPopup* |

2. Bind the **To_Button** pushbutton to this action:

| UI Element Name | Event Name | Action |
|---|---|---|
| **To_Button** | *onAction* | *ShowAddressbookPopup* |

3. To open the Web Dynpro window with the AddressbookView, open the implementation calls *AddressbookView.java* and add the following program code to the action handler `onActionShowAddressbookPopup()` to call the `showAddressbookPopup()` method on the Component Controller:

**EmailView.java**

```
...
public void onActionShowAddressbookPopup(
         com.sap.tc.webdynpro.progmodel.api.IWDCustomEvent wdEvent
)
{
   //@@begin onActionShowAddressbookPopup(ServerEvent)
   wdThis.wdGetPopupCompController().showAddressbookPopup();
   //@@end
}
```

```
...
```

## Creating and Implementing the Action and event handler *AddressSelected* on the AddressbookView View Controller

You now create and implement the action *AddressSelected* in the *AddressbookView*. This triggers the *AddressSelectedEvent* to the *EmailView*:

1. Open the controller of the *AddressbookView* (*PopupComp* → *Views* → *AddressbookView*).
2. Switch to the *Actions* tab page.
3. Create the action **`AddressSelected`**:

| Name of the Action | Validation | Event Handler |
|---|---|---|
| AddressSelected | checked | *onActionAddressSelected* |

4. Switch to the *Layout* tab page.
5. Bind the **ToolBarButton** that you created earlier to the action **AddressSelected**.
6. Switch to the *Implementation* tab page and add the following program code to the action handler `onActionAddressSelected()`:

**AddressbookView.java**

```
...
public void onActionAddressSelected(
    com.sap.tc.webdynpro.progmodel.api.IWDCustomEvent wdEvent )
{
    //@@begin onActionAddressSelected(ServerEvent)
    wdThis.wdGetPopupCompController().hideAddressbookPopup();

    //@@end
}
...
```

This sends the event *AddressSelectedEvent* of the *Component Controller* to the *EmailView*.

## Result

You have created a Web Dynpro window that embeds and implements a Web Dynpro view, which displays an address book.

### Next step:

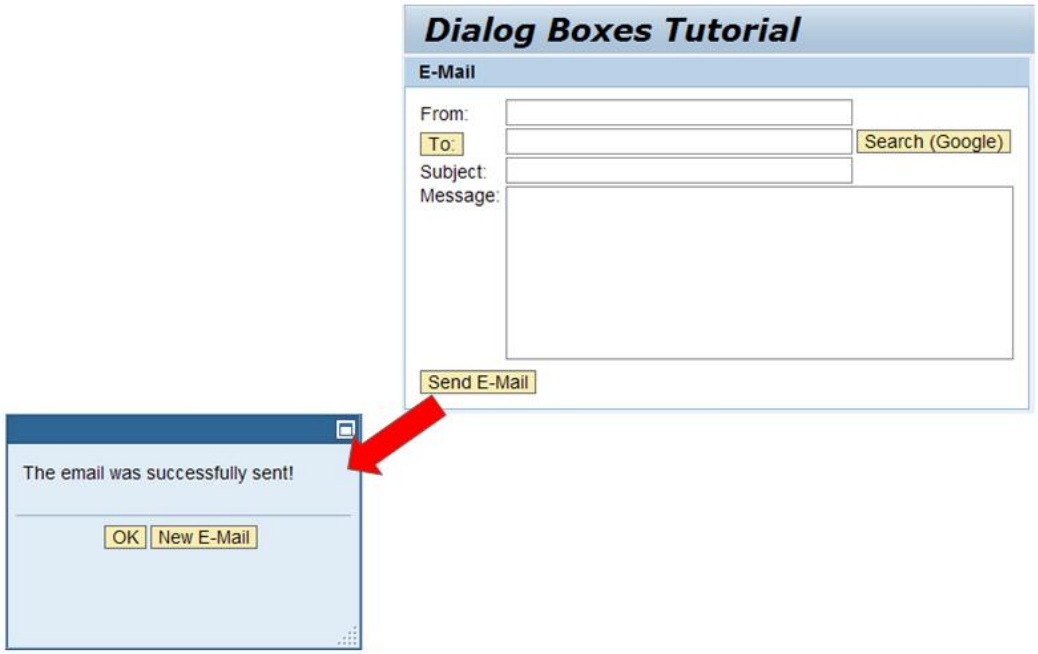
Creating a Confirmation Dialog Box [Page 22]

# Creating a Confirmation Dialog Box

You want a confirmation dialog box containing two pushbuttons to appear when *Send Email* is chosen. The *OK* pushbutton closes the window. The *new email* pushbutton deletes the content of the input form and closes the window.

- Confirmation dialog boxes are automatically destroyed by the Web Dynpro runtime when a button on them is clicked. This is due too their nature of only existing to get a single response from the user.

The functions for sending the e-mail are not implemented. For more information, see the tutorial “Using an E-Mail Web Service”.

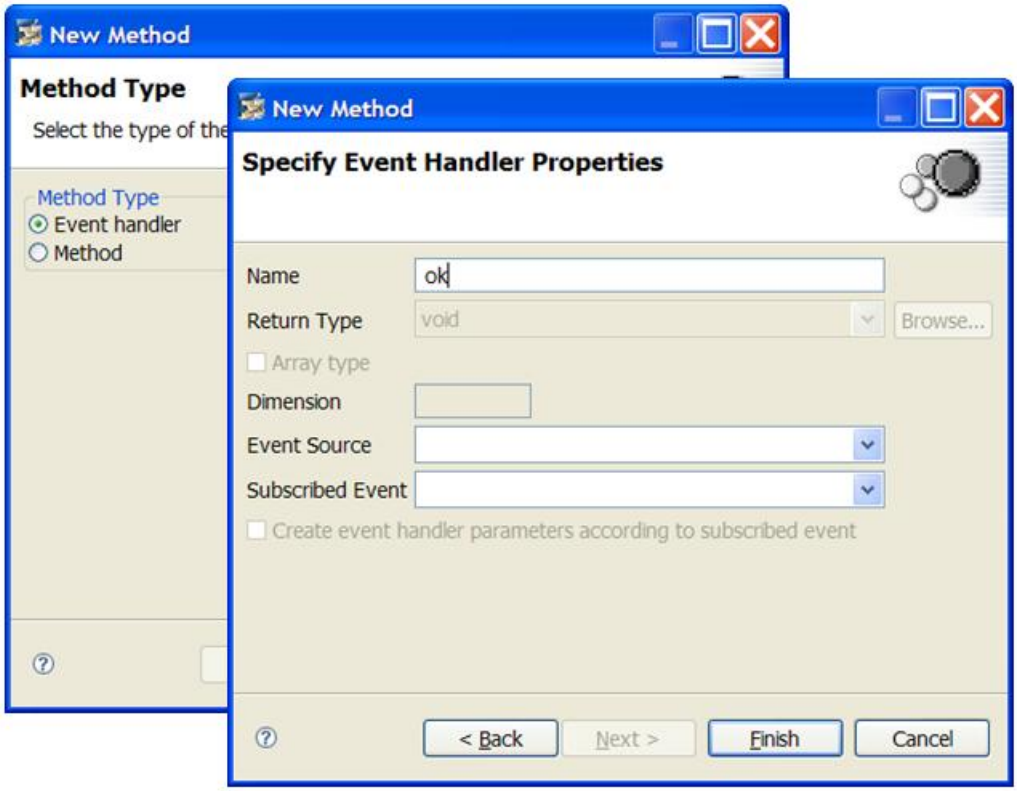



## Procedure

### Creating the Event Handlers *ok* and *new email*

1. Open the *EmailView* controller (*Web Dynpro Components* → *PopupComp* → *Views* → *EmailView*).
2. Choose the *Methods* tab page.
3. Choose *New*. In the wizard that appears, select *Event handler* and choose *Next* to confirm.
4. Enter the name **ok** and choose *Finish* to confirm.

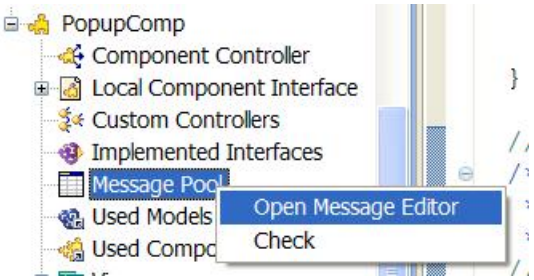
   The *Event Source* is assigned dynamically in the implementation.



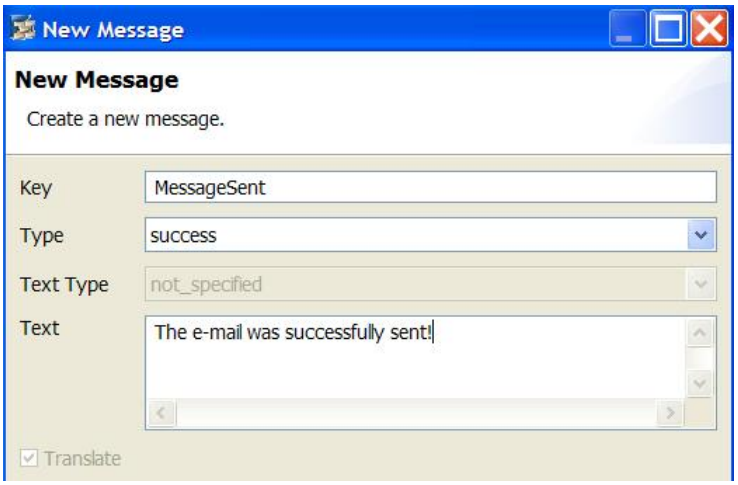

5. Use the same procedure as in steps 3 to 4 to create the event handler **`newEmail`**.

## Create a Successful Email Sent Message in the Message Pool

1. Open the Message Pool editor in the PopupComp Component.




2. Create a new message called *MessageSent* as shown below:



## Implementing the Event Handler *newEmail*

1. Switch to the *Implementation* tab page.
2. To clear the input fields in the form, reset the corresponding context attributes.

   Add the following program code to the method `newEmail()`:

**EmailView.java**

```
...
public void newEmail(
  com.sap.tc.webdynpro.progmodel.api.IWDCustomEvent wdEvent )
{
   //@@begin newEmail(ServerEvent)
   wdContext.currentEmailSettingsElement().setToAddress("");
   wdContext.currentEmailSettingsElement().setFromAddress("");
   wdContext.currentEmailSettingsElement().setMsgBody("");
   wdContext.currentEmailSettingsElement().setASubject("");
   //@@end
}
...
```

## Implementing the Action Handler *onActionSendEmail*

1. Add the following program code to the method `onActionSendEmail()`:

**onActionSendEmail()**

```
public void onActionSendEmail(
  com.sap.tc.webdynpro.progmodel.api.IWDCustomEvent wdEvent )
{
  //@@begin onActionSendEmail(ServerEvent)
   String dialogText = wdComponentAPI.getTextAccessor()
         .getText(IMessagePopupComp.MESSAGE_SENT);

   IWDConfirmationDialog dialog = wdComponentAPI.getWindowManager()
         .createConfirmationWindow(dialogText,
               IPrivateEmailView.WD_EVENTHANDLER_OK, "OK");

   dialog.addChoice(IPrivateEmailView.WD_EVENTHANDLER_NEW_EMAIL,
               "New E-Mail");
   dialog.show();
  //@@end
}
```

You create the confirmation window in the window manager by using `IWDWindowManager.createConfirmationWindow(String text,IWDEventHandlerId eventhandlerId,String buttonLabel).` The event handler *ok* that you defined in the view controller is transferred as the event handler.

To add a second pushbutton to the confirmation window, use `IWDConfirmationDialog.addChoice(IWDEventHandlerId eventhandlerId, String buttonLabel).` The event handler *new email* that you defined earlier is transferred as the event handler.

The methods `destroyInstance` and `hide` are deprecated on the IWDConfirmationDialog class since the Web Dynpro runtime takes care of this for you.

## Next Step:

Executing the Application [Page 25]

Executing the Complete Application

Once you reach this stage, you have completed the development of the Web Dynpro application. You can now start it in the Web browser, as described below.

## Prerequisites

- ☐ The SAP AS Java has been started.

## Procedure

1. Save all the files that have been modified.

   *Development Component Build*

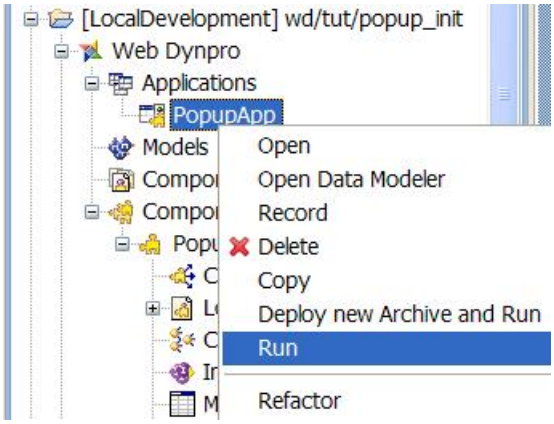
2. In the Web Dynpro Explorer, in the context menu of the project node (**`wd/tut/popup_init`**), choose *Development Component* > *Build…*

   Deploy

3. In the Web Dynpro Explorer, in the context menu of the project node (**`wd/tut/popup_init`**), choose *Deploy*.

   Run

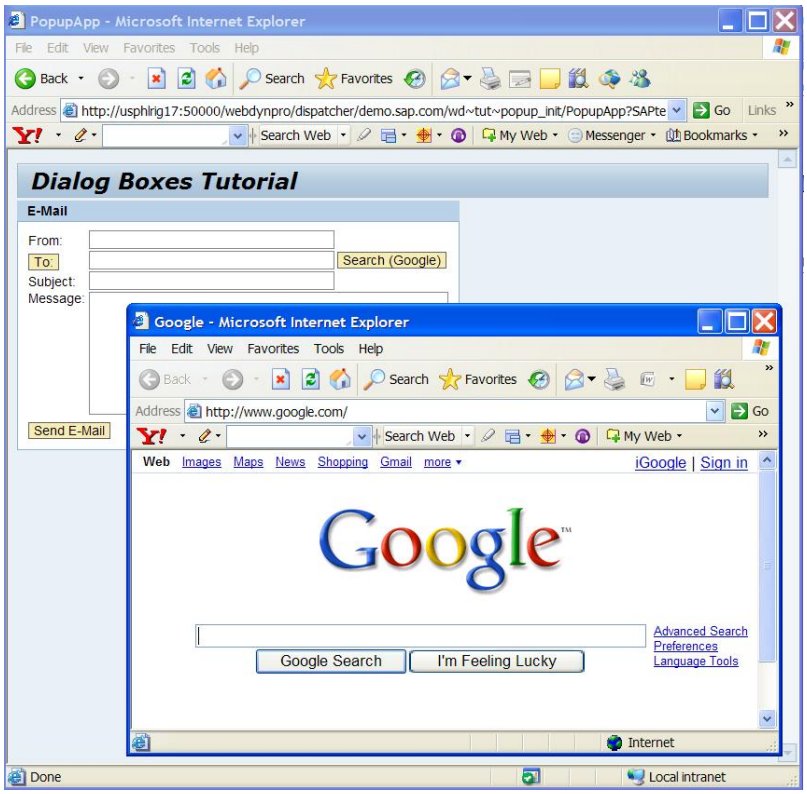
4. In the Web Dynpro Explorer, in the context menu of the Application node (**PopupApp**), choose *Run*.



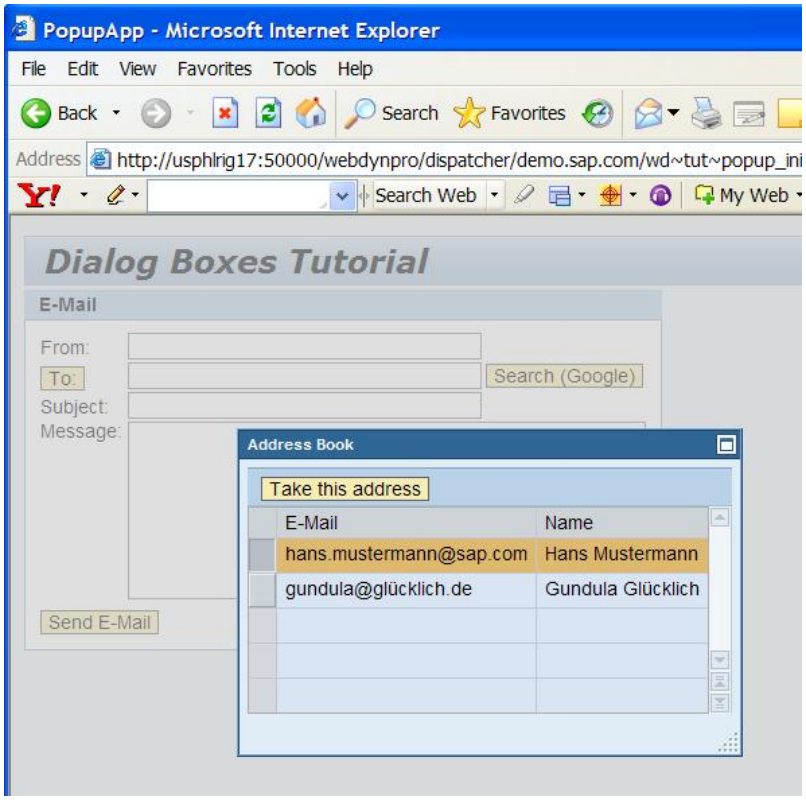

## Result

The Development Component project is built, deployed and run.

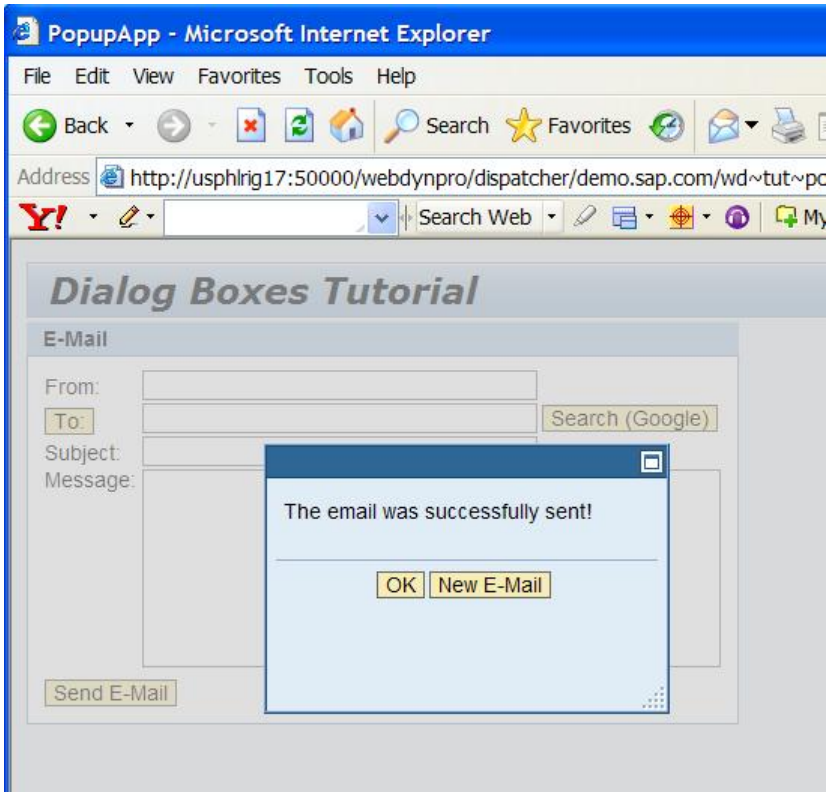
When you choose *Search (Google)*, an external window displaying the Google homepage appears.




When you choose *To*, a dialog box displaying the address book appears.




If you complete the e-mail form and choose *Send Email*, the confirmation window appears.


PopupApp - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://usphlrig17:50000/webdynpro/dispatcher/demo.sap.com/wd~tut~pc
Search Web
Dialog Boxes Tutorial
E-Mail
From:
To:
Search (Google)
Subject:
Message:
The email was successfully sent!
OK
New E-Mail
Send E-Mail