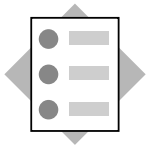
Web Dynpro Windows

Contents:

- **Default Windows**
- **Popup Windows, External Windows, Confirmation Dialog Windows.**
- **Create Windows.**
- **Web Dynpro Windows API.**

Web Dynpro Windows: Objectives



After completing this lesson, you will be able to:

- **Understand the different windows you can create in your Web Dynpro applications.**
- **Create new windows using the SAP NetWeaver Developer Studio.**
- **Create different kinds of popup windows using the Web Dynpro Windows API.**

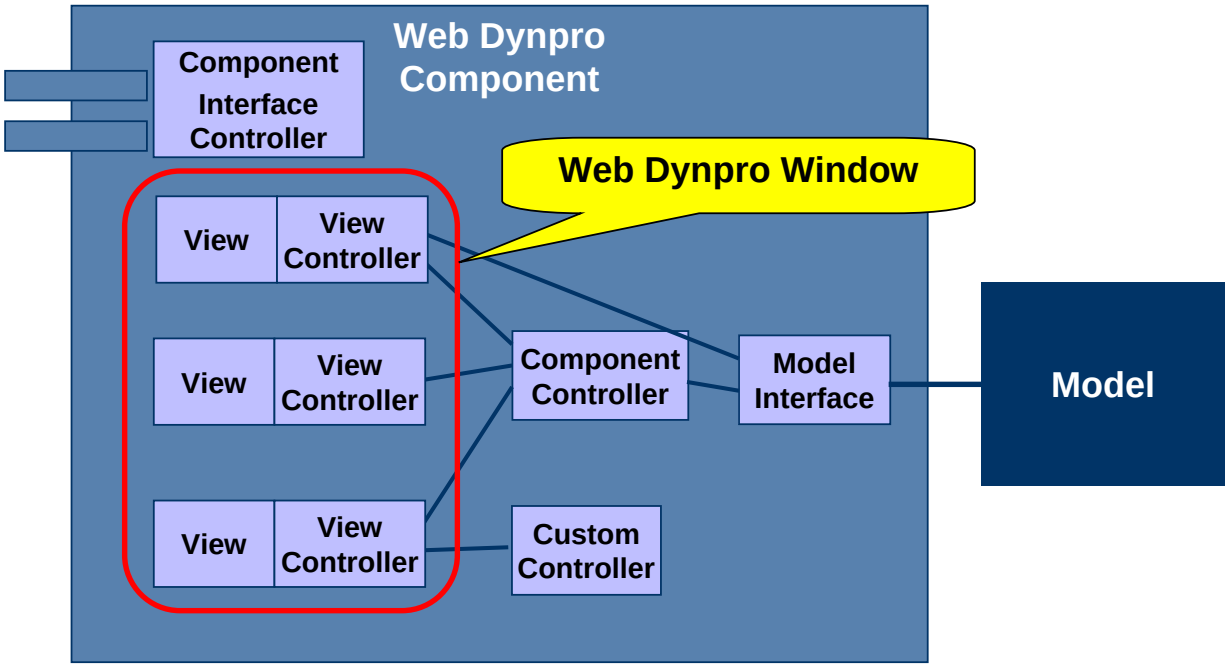
Web Dynpro Windows

Web Dynpro is a very robust programming environment that has support for multiple window types:

- Default Windows – the main window of your application.
- Popup Windows – windows that you can programmatically “popup” and display to your users.
- External Windows – windows that show up in a separate browser.
- Confirmation Dialog Windows – windows that ask a question or give information that your users must respond to.

- **Web Dynpro Windows**
  The different kinds of Web Dynpro windows.

Web Dynpro Windows

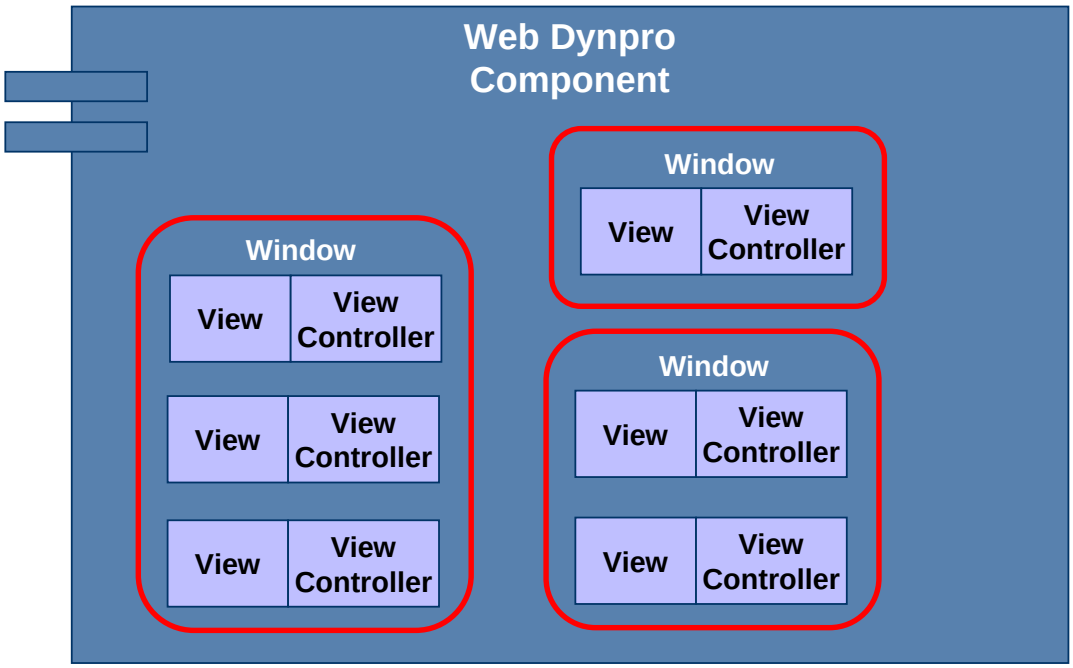


Each Component by default contains a Window, in turn the Windows contain the Views

■ **Web Dynpro Windows**

As we can see here, Web Dynpro Components are made up of views and controllers.
Web Dynpro Windows are made up of Views, and View Sets.
All visual elements that a user of a Web Dynpro application sees belong to a Window.

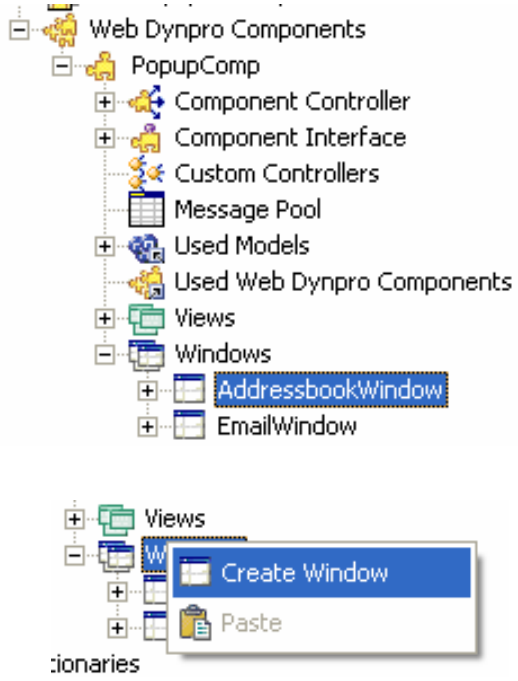
Multiple Windows

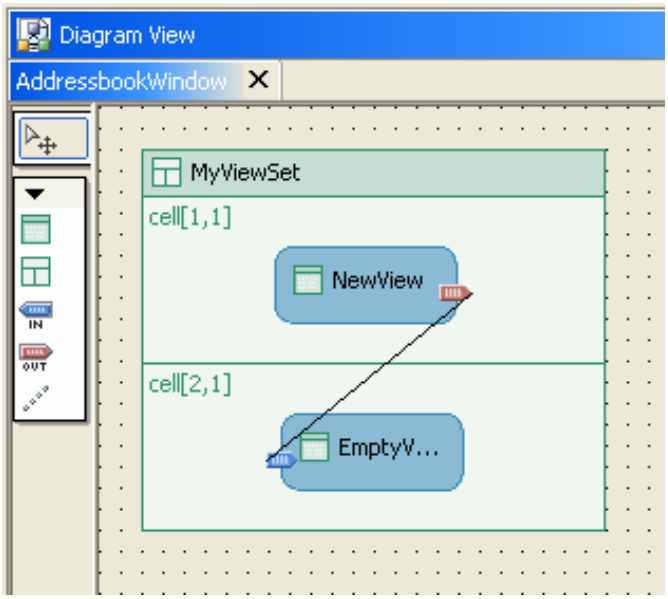


Web Dynpro Components Can Contain Multiple Windows.

- **Multiple Windows**
  Web Dynpro components can have multiple windows.
  Some a single view can belong to any number of windows.

Windows – SAP NetWeaver Developer Studio





■ **Windows – SAP NetWeaver Developer Studio**

To create a window you just right click on the “Windows” node of a Web Dynpro project and select “Create Window”.

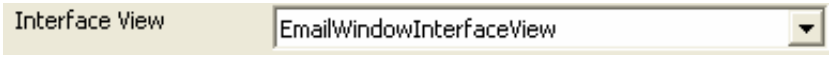
You can see a graphical view of your window by double clicking on it. This will display the “Diagram View” as is shown on the above right.

You can use the “Diagram View” to create and add views, plugs, navigation links, and viewsets.

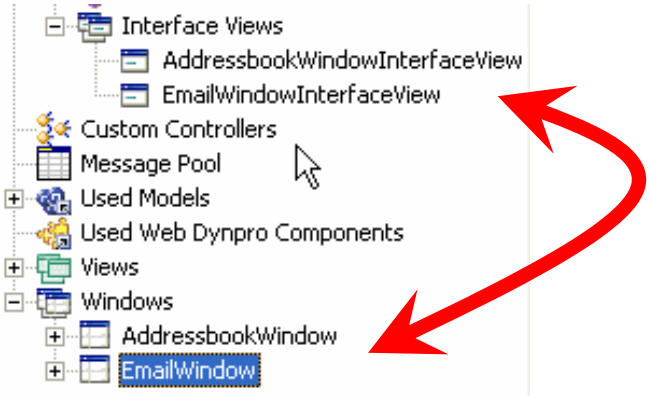
Default Window

Web Dynpro Default Windows

- The Window that exists in the Browser at application startup
- The application selects its default window Interface View:



- For each window a Interface View is created

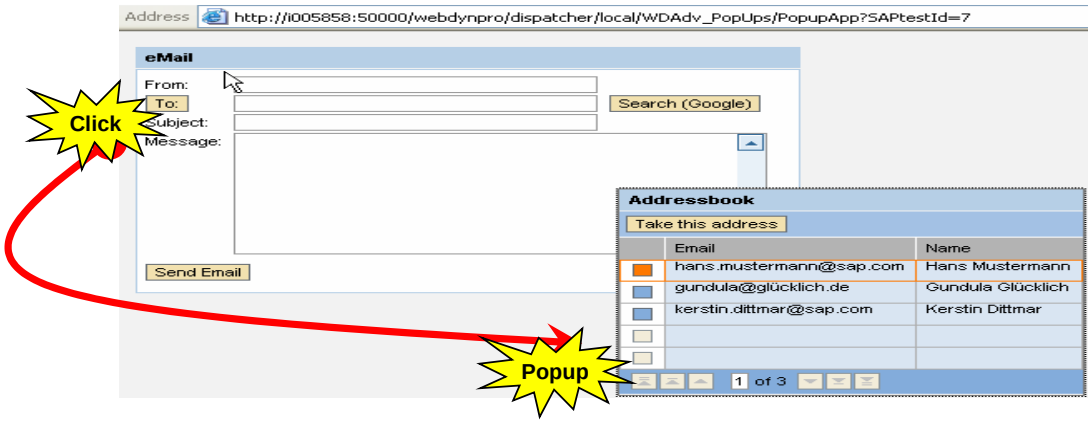


- **Default Window**
  When creating an application you must select a default window. This is done by setting the “Interface View” property of a Web Dynpro application.
  Each Window has an associated Interface View as shown above.

Popup Window

Pop-Up Windows

- Windows that appear as the result of some action (ex: clicking on a button).
- Exists in the same browser screen that created them.



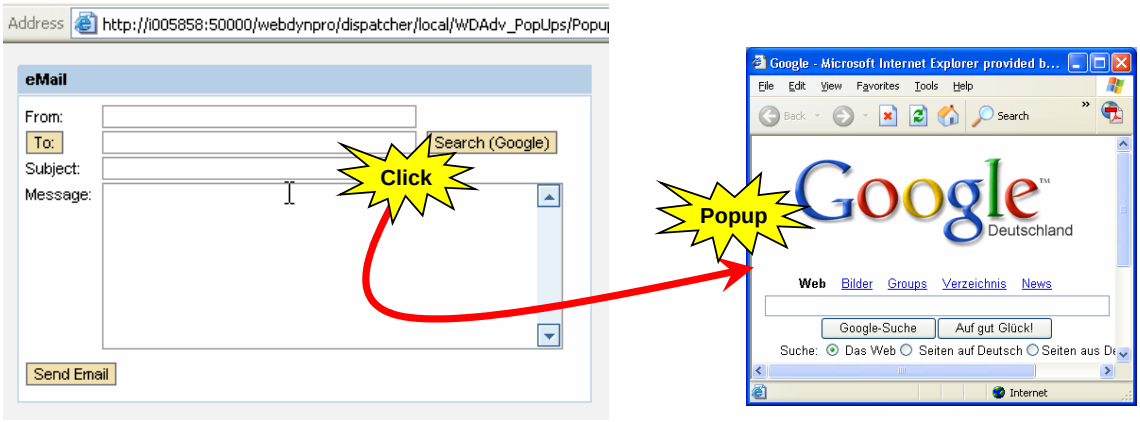
- Popup windows can *not* be resized or moved by the client.
- Currently only modal windows are supported.

- **Popup Window**

  Popup windows are based on dynamic html and thus in the client's browser they are part of the existing "page" that started the popup window.
  Must use the Windows API to create the Popup...this will be shown in later slides.
  A declarative approach for embedding popup windows into Web Dynpro View-Compostions is not supported yet.
  Must create a Web Dynpro Window in your component to hold the UI content of your popup window.
  Currently only modal windows are supported.
  A modal window means that users can only interact with the popup window when it is active. The example above shows a popup window, as long as it is active (i.e. – is viewable), the user can not work on the main window.

External Popup Windows

- Windows that appear as the result of some action (ex: clicking on a button).
- Exist a separate browser window.



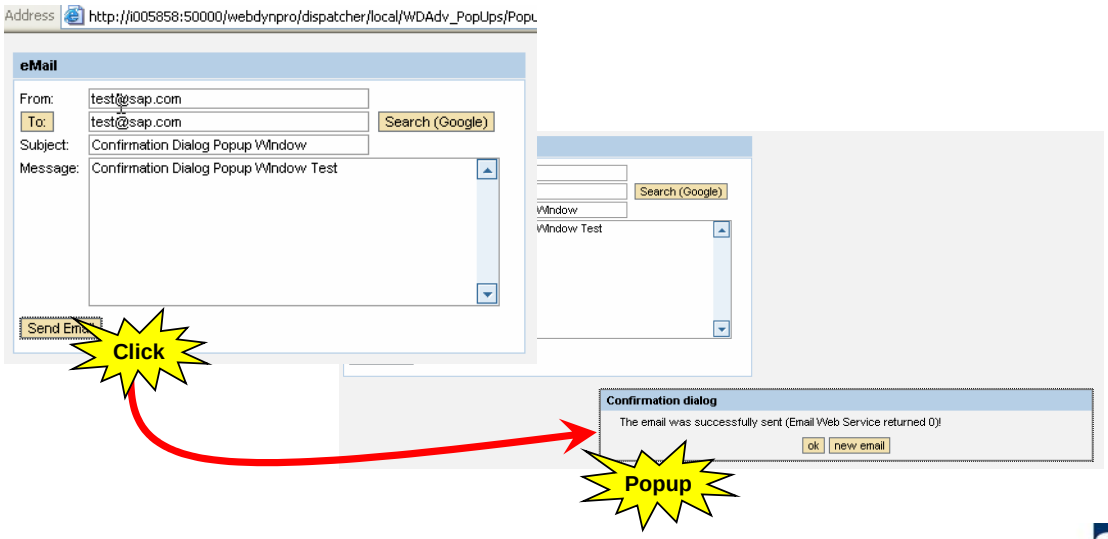
- Currently only non-modal windows are supported.

- **External Popup Window**
  Window API used to create popup…will be shown later. Currently only non-modal windows are supported. Non-modal windows are windows that are popped up but users can continue to work with the main window and not have to deal with the non-modal window.
  Since the external window is in its own browser, user are free to do with it what they want…close it, go to another site, an so on.

Confirmation Dialog Popup Window

Confirmation Dialog Popup Windows

- Windows that appear as the result of some action (ex: clicking on a button).
- Exists in the same browser screen that created them.



- **Confirmation Dialog Popup Window**
  Confirmation Dialog windows can not be moved or resized by the client.
  Can add multiple “choice” buttons to your dialog. Each choice is assigned to an event handler.
  Confirmation Dialogs are generic popup-windows containing a confirmation text and a set of “choice” buttons. The button-clicks are handled by the defined event handlers (in same component).
  Confirmation Dialogs are always modal windows.

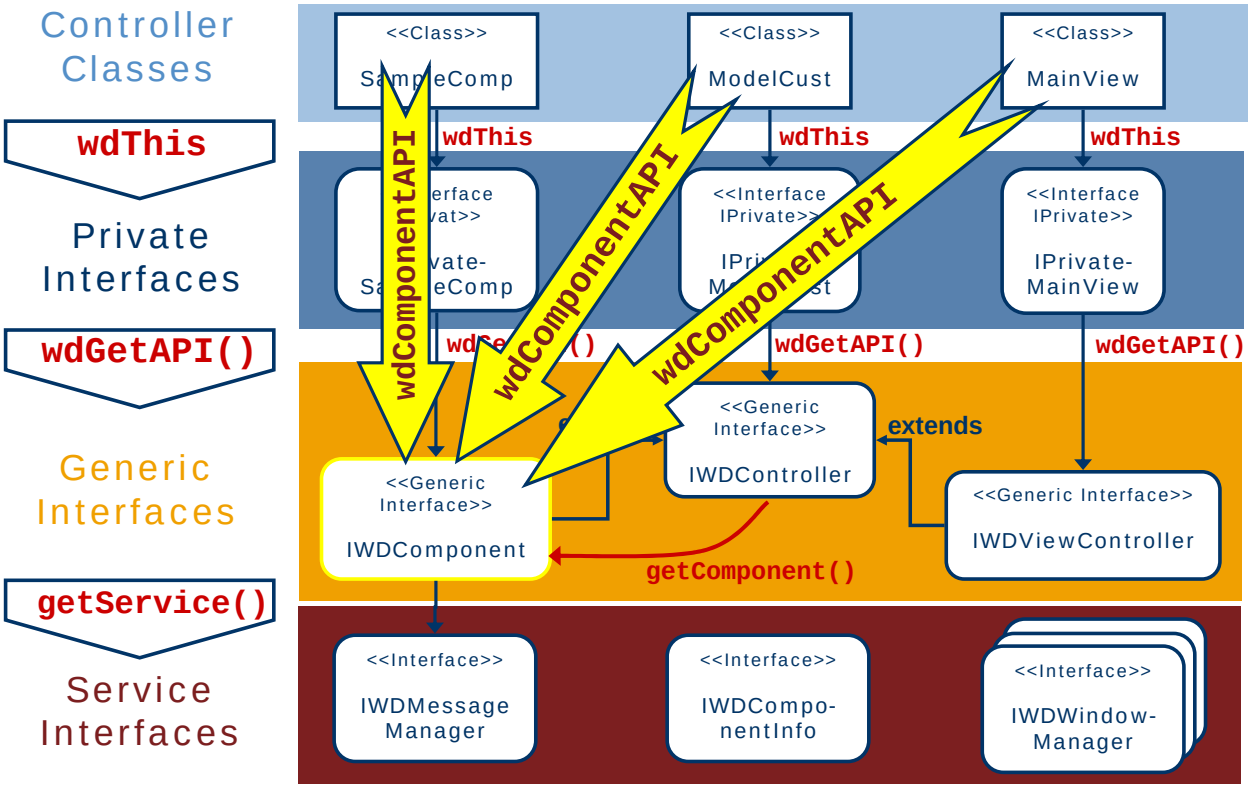
Web Dynpro Foundation Framework Window API

The Web Dynpro Foundation Framework provides the following Interfaces for programmatically embedding popup windows inside view-compositions.

- IWDWindowManager – Used to create windows.
  - Methods: createWindow(…), createConfirmationWindow(…), createExternalWindow(…).
- IWDWindowInfo – Definition of a window. Needed when creating a Popup window.
- IWDWindow – Interface of a created window that can be displayed to the client as a popup.
- IWDConfirmationDialog (extends IWDWindow) – Interface of a created Confirmation Dialog that can be displayed to the client as a popup.

- **Web Dynpro Foundation Framework Window API**
  Above are the Window Interfaces of the Web Dynpro Foundation Framework…if working with window popups, you will need to learn these Interfaces very will. The following slides go into the use of these interfaces.

Accessing Service Interfaces from inside Controllers



■ **Accessing Service Interfaces from inside Controllers**

To access the Window Manager (IWDWindowManager) service, you need to go through the component's API interface.

Use the "shortcut" variable defined in all controllers to do this:

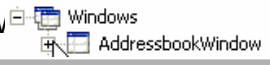
wdComponentAPI.getWindowManager() – this is a lot better than have to type wdThis.wdGetAPI().getComponent().getWindowManager();

In the constructor of all controllers you will see wdComponentAPI = wdThis.wdGetAPI().getComponent();

Popup Window Programming: Create, Position, Open

Window Creation

- Assumes that the window “AddressbookWindow” is already defined as one of your components windo



Example

```
//Get the WindowInfo for AddressbookWindow
IWDWindowInfo windowInfo = wdComponentAPI.getComponentInfo()
      .findInWindows("AddressbookWindow");
//Get the WindowInfo for AddressbookWindow
IWDWindow window = wdComponentAPI.getWindowManager()
      .createWindow(windowInfo, true);
```

Window Positioning and Opening.

- **Positioning involves where the popup window will show up in the browser when it is opened.**

Example

```
window.setWindowPosition(300, 150);
// or could use:
// window.setWindowPosition(WDWindowPos.CENTER);
window.open();
```


- **Popup Window Programming: Create, Position, Open**

  To create a popup window you first need to get the Window interface definition (IWDWindowInfo). In the example above you can see this done for the window “AddressBookWindow”.
  Once you have the Window Info Interface you can call the Window Manager to create the window (IWDWindow).

  In the method createWindow(windowInfo, true) -> the second parameter is a boolean representing if the created window should be modal or not…currently only modal windows are supported! In the future you can set this to false to get an non-modal window. Currently though, setting it true or false changes nothing.
  You can position your popup window relative to the upper left hand corner of the web dynpro shown in the browser. The above shows the method setWindowPosition(300, 150) – these represent pixels.
  You can also use the class WDWindowPos constants to set the position. Example window.setWindowPosition(WDWindowPos.CENTER).

Popup Window Programming: Preservation, Close, Destroy

Window Preservation

- After the window is created, it should be stored in a controller context variable, or a implementation member variable of type IWDWindow.
- It is necessary to preserve the window, since it will need to be closed and/or destroyed when the client is done with it.

```
wdContext.currentPopupElement().setWindowInstance(window);
// or assuming you have create a member variable
// IWDWindow window = null; You could do the following:
// this.localWindow = window;
```

Window Closing & Destroying

- If an event is triggered which should close your popup window you can use the close() method. This method though does not remove the window, and you can use the window again if you need to.
- If you have closed your window and you are no longer in need of it, then you should call the destroy() method so it can be removed from memory.

```
window.close();
window.destroy();
```


- **Popup Window Programming: Preservation, Close, Destroy**

It is key that you save your window in a context attribute or a member variable of your controller. This is because even though you create a window, you need to keep a reference to it for when you need to close the window.
Example: A user clicks a button that calls an event on a view controller. The event creates a popup window. The user then interacts with your popup window then clicks a button on it that calls an event that in turns accesses the reference to the window and calls its close() method.

Above shows an example of saving the window in a context element, but you can also save it in a member variable. To do this you have to go to the bottom of your controller where there is a special spot to create member variables.

If you want to close your window but access it again later on, you can use the close() method. But remember this method will not release the window object to be garbage collected!
If you want to close your window and have it release to be garbage collected you should use the destroy() method. If you want to use the window again you will have to recreate it.
Always remember to destroy() your windows when you are done with them!

External Popup Window Programming

Creating External Popup Windows

- Can launch an external browser, that will open a URL that you pass it.
- Once created, it is up to the user to control.

Example

```
public void onActionShowGoogleWindow(...)
 {
   //@@begin onActionShowGoogleWindow(ServerEvent)
        IWDWindow window = wdComponentAPI.getWindowManager()
                .createExternalWindow("http://www.google.com",
                                      "Google!", true);
        window.open();
   //@@end
 }
```

- All the other Popup window APIs apply. Just creating is different.

External Popup Window Programming

When you create an external window you use the .createExternalWindow(…) method of the Window Manager.
createExternalWindow(String Url, String Title, boolean modal) -> only non-modal windows are currently supported!

Confirmation Dialog Window Programming - Create

Creating Confirmation Dialog Windows

- Confirmation Dialog windows allow you to create a simple window which states information or is asking a question (ie: Are you sure you want to delete?).
- Must map the buttons on the Confirmation Dialog to event handlers in your view controller.
- A confirmation dialog must have at least one button. The example below shows a confirmation dialog with an “ok” button being created

Example

```
dialogText = "Your data has been saved.";
//Get Event Handler Info
IWDEventHandlerInfo eventHandler = wdControllerAPI.getViewInfo().
    getViewController().findInEventHandlers("ok");

//Create the confirmation dialog, with an "ok" button
IWDConfirmationDialog dialog =
    wdComponentAPI.getWindowManager()
       .createConfirmationWindow(dialogText, eventHandler, "ok");
```

- All the other Popup window APIs apply.

- **Confirmation Dialog Window Programming - Create**

CreateConfirmationDialog(confirmationText, eventHandler, buttonLabel);

confirmationText – is the statement or question you want to convey to the user.

eventHandler – event for the default button.

buttonLable – label on the default button.

All confirmation dialogs are non-modal.
All confirmation dialogs need at least one choice/button, hence the reason the createConfirmationWindow(…) method requires an eventHandler!

wdControllerAPI.getViewInfo().getViewController().findInEventHandlers("ok"); -> gets a reference to the event handler called “ok”. This event handler must be defined as one of the methods of your controller. Method setIcon(String iconUrl) of IWDConfirmationDialg can be used to add an appropriate Icon to your confirmation dialog.

iconUrl - the absolute url for the icon.

Confirmation Dialog Window Programming – Add Choices

Adding Choices to Confirmation Dialogs

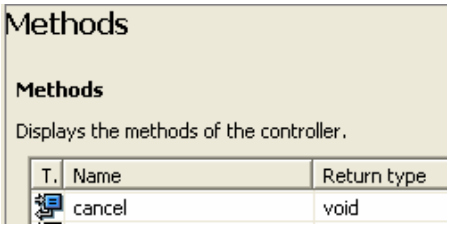
- Confirmation Dialogs can have multiple choice buttons.
- Each one must be mapped to an event handler. Event handler takes appropriate action.

Example

```
//Get Event Handler Info
IWDEventHandlerInfo cancelHandler =
    wdControllerAPI.getViewInfo().
        getViewController().findInEventHandlers("cancel");
//Add the choice Cancel to the dialog
dialog.addChoice(cancelHandler, "Cancel");
//Open the window
dialog.open();
```

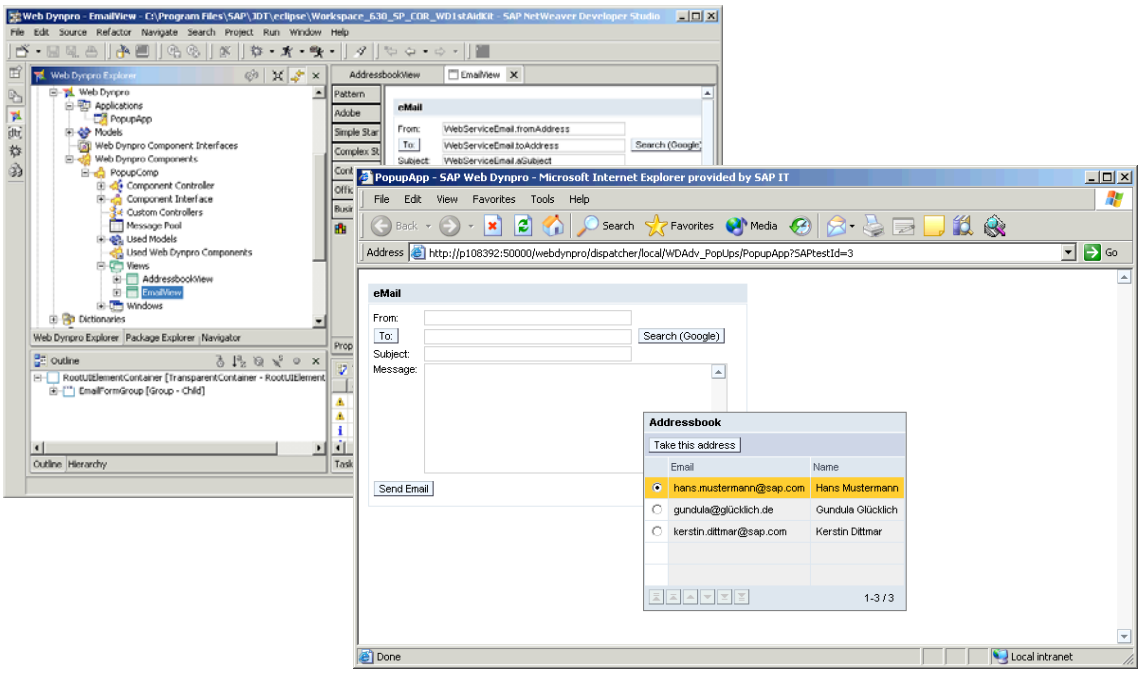
Event Handlers

- Should be created as one of the view's event handler methods.



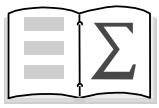
- **Confirmation Dialog Window Programming – Add Choices**
  You can add multiple "choice" buttons to your confirmation dialog.
  Can add disabled button with method addChoice(IWDEventHandler eventHandler, String buttonLabel, boolean enabled).

Example: Popup Windows



See running example ...

Web Dynpro Windows: Summary



You should now be able to:

- **Understand the different windows you can create in your Web Dynpro applications.**
- **Create new windows using the SAP NetWeaver Developer Studio.**
- **Create different kinds of popup windows using the Web Dynpro Windows API.**