How to Use Context Menus in a Web Dynpro for Java Application

Applies to:

Web Dynpro for Java 7.11. For more information, visit the Web Dynpro Java homepage.

Summary

This tutorial explains the Web Dynpro context menu programming model and demonstrates how to implement context menus in a file explorer application.

Author(s): Web Dynpro Java Team

Company: SAP AG

Created on: 29 June 2010

Table of Contents

Introduction

In this tutorial you will learn how to implement context menus in a Web Dynpro application. This wil include learning how to create context menus in the View Designer and how to assign them to areas of your screen. You will also learn how to use the view controller method `wdOnContextMenu` to create and modify context menu entries programmatically.

Prerequisites

Systems, Installed Applications, and Authorizations

You need the NetWeaver Developer Studio (Version 7.11 or later) to compile and deploy the tutorial application. The application server used should have the same version as the NWDS or a newer version.

The tutorial application is available as a development component (DC). You need to import the software component **HM-WDUIDMKTCNT**, which contains the DC **tc/wd/tut/menu/ctx**. The exact steps are described in a separate document.

Objectives

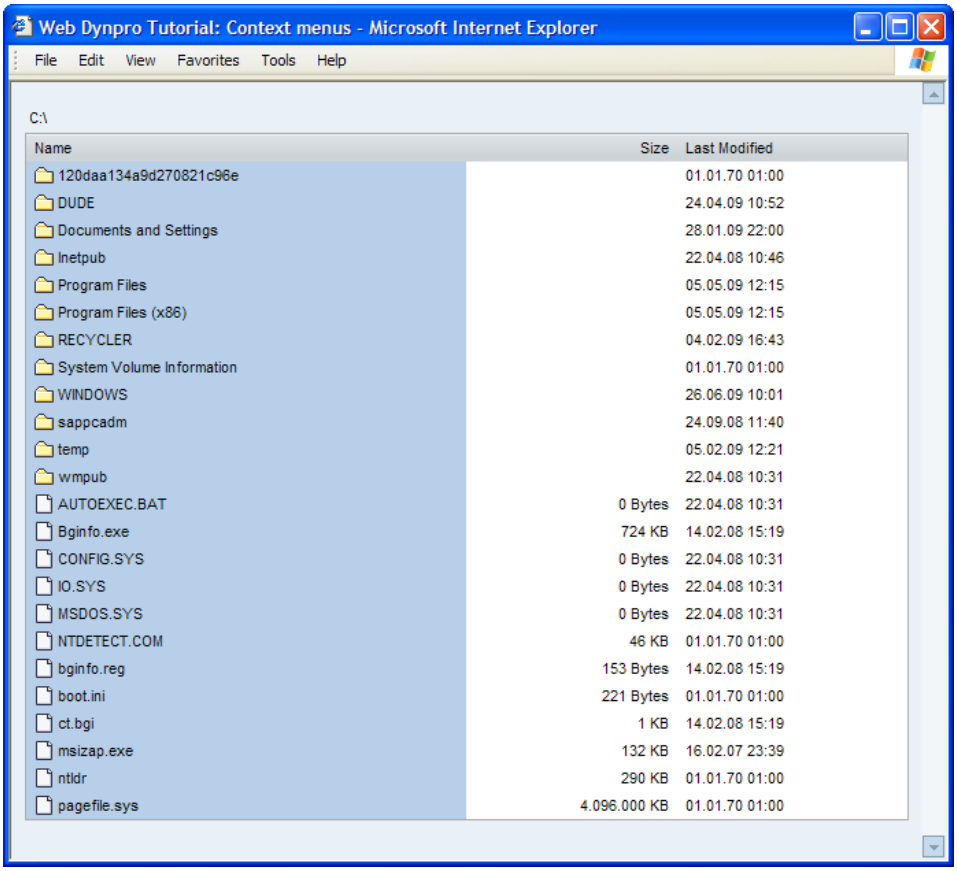
After working through this tutorial you should be able to:

- Understand the programming model for context menus in Web Dynpro
- Use the NWDS View Designer to create context menus and assign them to view areas
- Add code to the view controller hook method `wdOnContextMenu()` to create and modify context menu entries.

The Tutorial Scenario: A File Browser

The tutorial implements a file system browser (on the server) that offers some of the context menus you probably know from Windows Explorer.

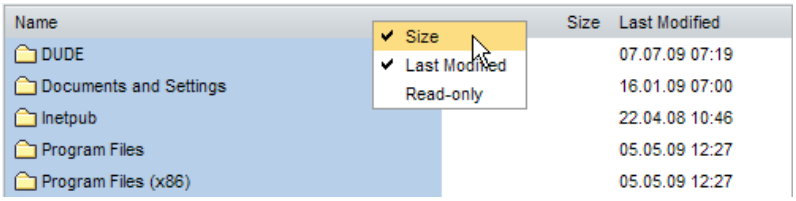
When you start the tutorial application, a screen similar to the following one will appear:



Initially, the root directory of the server's file system is displayed. Let's have a look at the context menus available in this application.

Column Display Context Menu

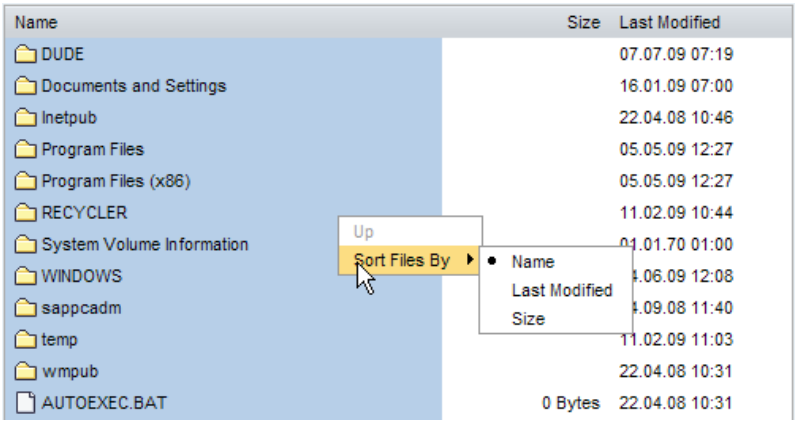
Using the right mouse button, click on any of the table column headers. A context menu for showing/hiding the columns appears. (There is no menu entry for the "Name" column because this column is always visible.):



Check/uncheck the entries in the context menu and note the changes in the table.

Navigation and Sorting Context Menu

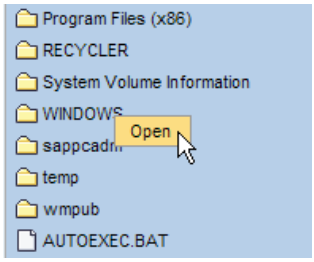
Do a right-click anywhere inside the table except over the file or directory names. A context menu with entries for sorting the table and navigating upwards in the file system appears:



The *Up* entry is disabled if the folder currently displayed has no parent folder.

File-Specific Context Menu

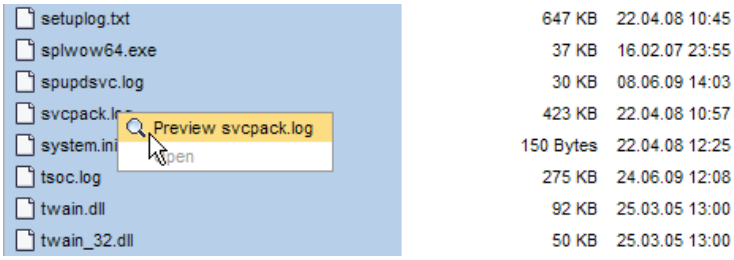
Do a right-click on a directory name or icon. A context menu with an entry for opening the directory appears:



When you select the menu entry, the files in the target directory are displayed in the table.

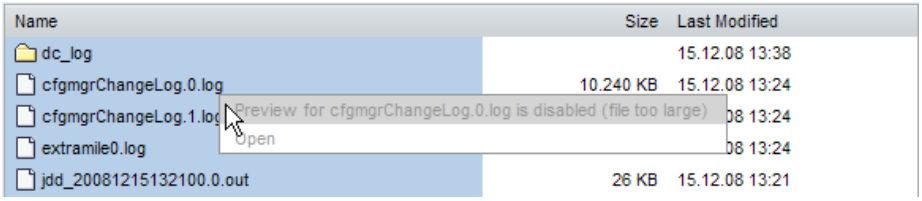
Navigate to any directory containing text files, for example, log files with the extension ".log" (for the exact list of file types, see the standard expression PATTERN_TEXT_FILE in the view controller implementation).

Do a right-click on such a file. An additional context menu entry for opening a preview of the file appears:



When you select the *Preview <filename>* entry, a popup window is opened and the content of the file is displayed.

Note that the context menu entry appears disabled for large files (more than 512 Kbytes):

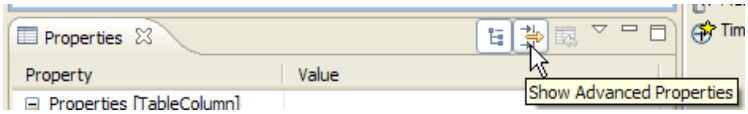


Assigning Context Menus to Screen Areas

The Web Dynpro context menu programming model provides a way to assign context menus to screen areas efficiently. Every UI element that supports context menus has two properties: `contextMenuId` and `contextMenuBehaviour`.

IMPORTANT:

In the 7.11 IDE, the `contextMenuId` and `contextMenuBehaviour` properties are only visible if *Show Advanced Properties* is selected:



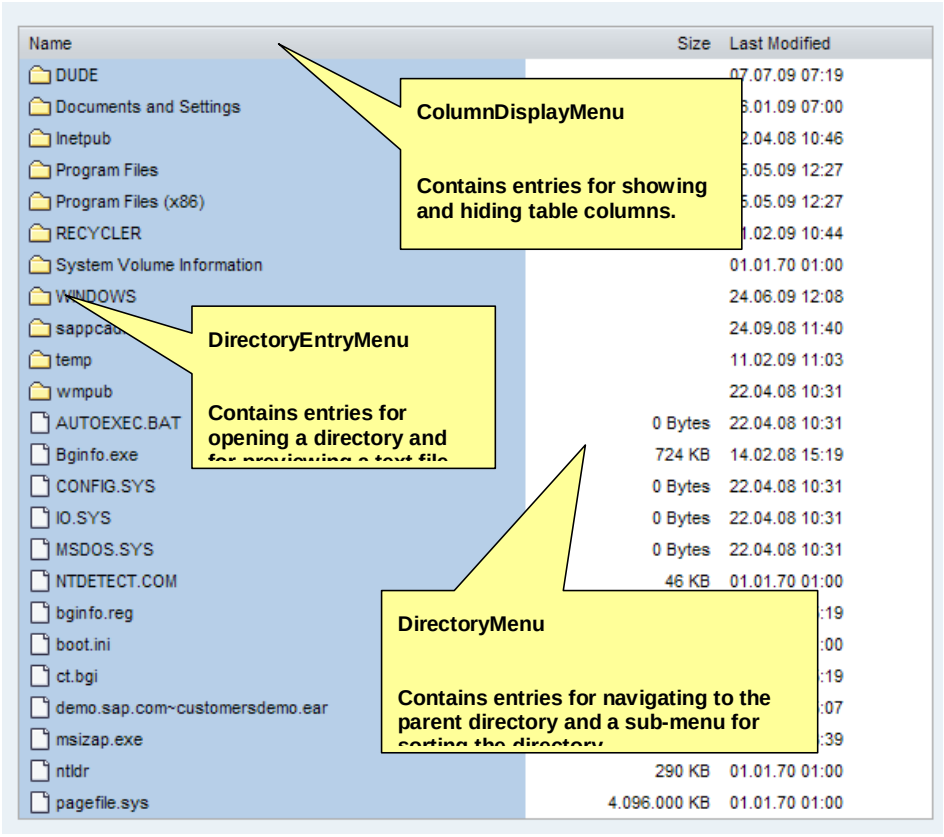
The `contextMenuId` property is used to assign a context menu to a UI element. The `contextMenuBehaviour` property defines how a UI element behaves with respect to the context menu defined by its parent in the UI element hierarchy.

The `contextMenuBehaviour` property can have the following values:

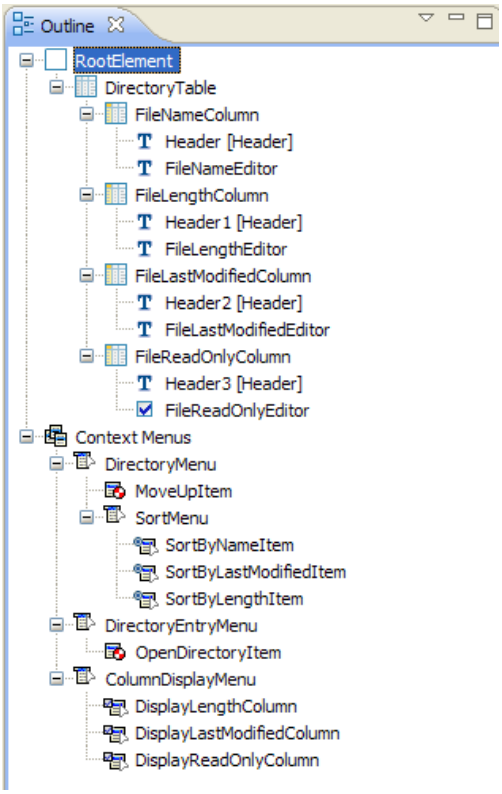
- `INHERIT`: the UI element inherits the context menu defined by its parent (or ancestor). This is also the default value for that property, which has the effect that context menu definitions automatically propagate downwards in the layout hierarchy.
- `SUPPRESS`: the UI element suppresses the context menu defined by its parent. The context menu is also suppressed for the children of the UI element. Using `SUPPRESS`, an inherited context menu definition can be suppressed for a complete sub-tree of the layout hierarchy.
- `PROVIDE`: the UI element provides the context menu given by the value of the `contextMenuId` property.

Comment [D1]: Vielleicht auch „or overwrites parent's behaviour"?

In our example application, we want to assign context menus to the different areas in the table as follows:



To define the context menu assignments correctly, it is important to understand the hierarchy of the Web Dynpro layout for this screen:



First, the “DirectoryMenu” context menu is assigned to the “DirectoryTable” UI element. The effect is that this context menu appears everywhere inside the table (a consequence of the default value `INHERIT` for the context menu behavior property).

We assign:

DirectoryTable => (`PROVIDE`, “DirectoryMenu”)

Next, we want the “ColumnDisplayMenu” context menu to appear above every column header. We define:

*Column => (`PROVIDE`, “ColumnDisplayMenu”)

As a result, this context menu appears inside the complete table header (not only on the column header text).

The other consequence is that the “ColumnDisplayMenu” now also appears on the cell editors of the table columns (but not in the free area inside the table body). To display the “DirectoryMenu” on the cell editors instead, we define:

*Editor => (`PROVIDE`, “DirectoryMenu”)

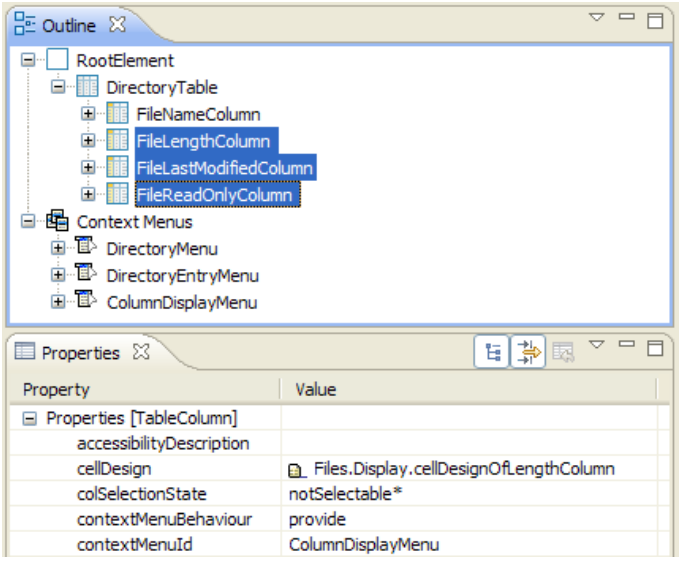
For the editor of the *Name* column, we want the *DirectoryEntryMenu* to appear. Therefore, we define:

FileNameEditor => (`PROVIDE`, “DirectoryEntryMenu”)

This should give you the idea behind the properties `contextMenuId` and `contextMenuBehaviour`.

These assignments can also be done at runtime using the UI element API. We will not describe the creation and assignment of context menus through the API in this tutorial. We will, however, describe how to modify context menus defined and assigned at design-time using the runtime API.

It should now be clear how the definitions above are created using the View Designer. Note that you can select multiple UI elements in the *Outline* view and set common property values for all of them in one go:



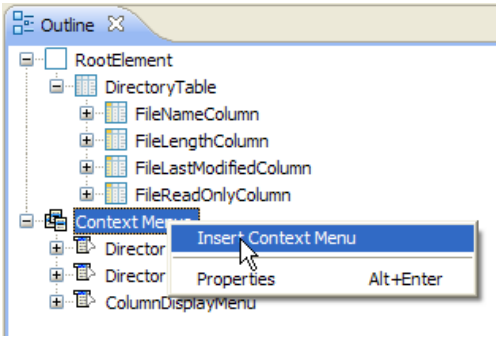
Creating Context Menus and Assigning Actions

Context menus are created in the View Designer in the same way as other view elements. The only difference is that they are stored in a different collection inside their view.

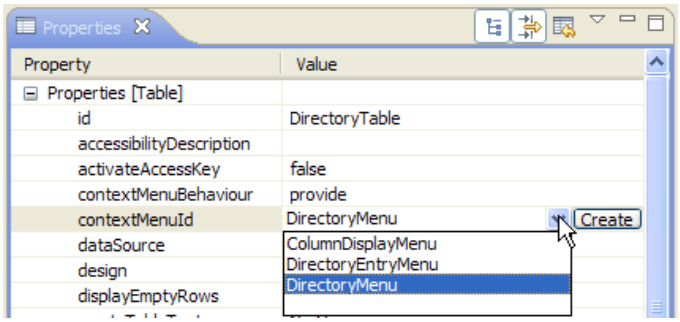
This is important to know if you want to create context menus using the API. You **must** use the context menu-specific API `IWDContextMenuManager` and the hook method `wdOnContextMenu()` to create context menus. Do not use the `IWDView` API and hook method `wdDoModifyView()`!

Let us have a look at how the context menus defined above are created in the View Designer:

There is a node labeled “Context Menus” in the “Outline” view. Right-click this node to create and add context menus:

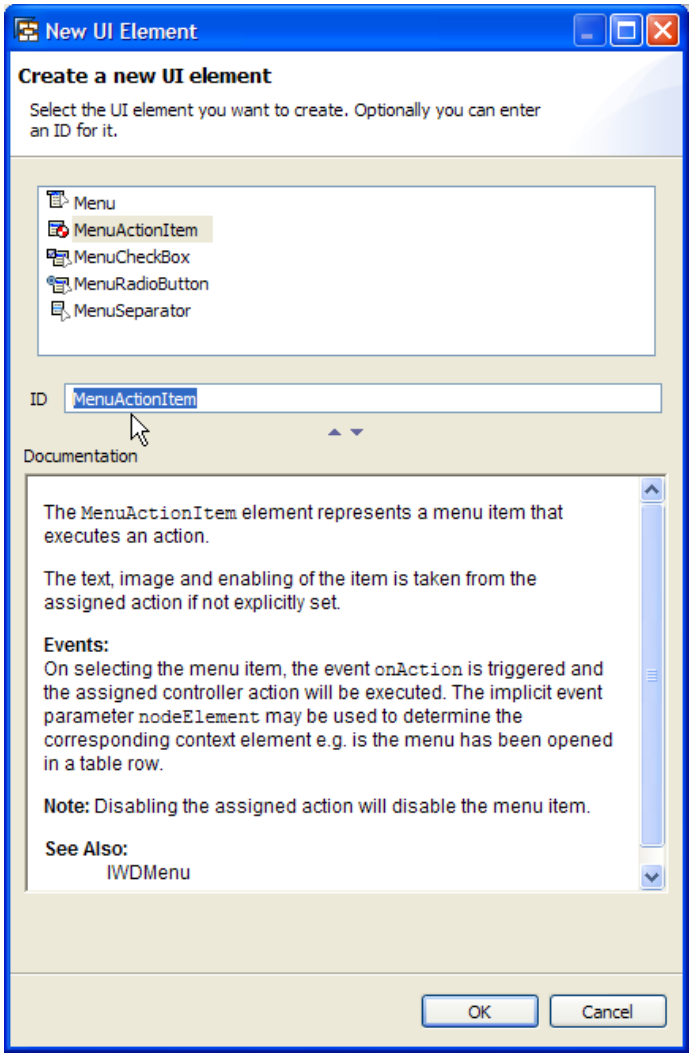


A context menu created in this way is available in the drop-down list for setting the `contextMenuId` property:

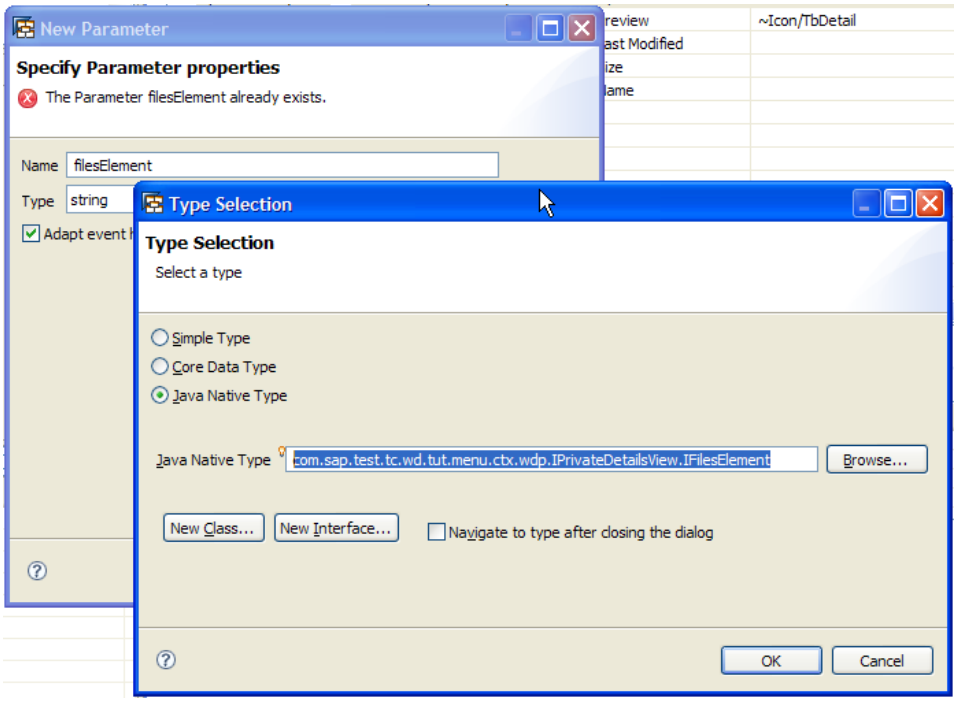


You can also use the “Create” button shown above. In this case, a new context menu with a generic ID such as “Context Menu1” is created (you should change the ID to something more appropriate afterwards.)

The creation of context menu **items** works exactly as for menus created in the view layout. The same set of menu items (`MenuActionItem`, `MenuCheckBox`. `MenuRadioButton`, `MenuSeparator`, `Menu`) is available for context menus as well:

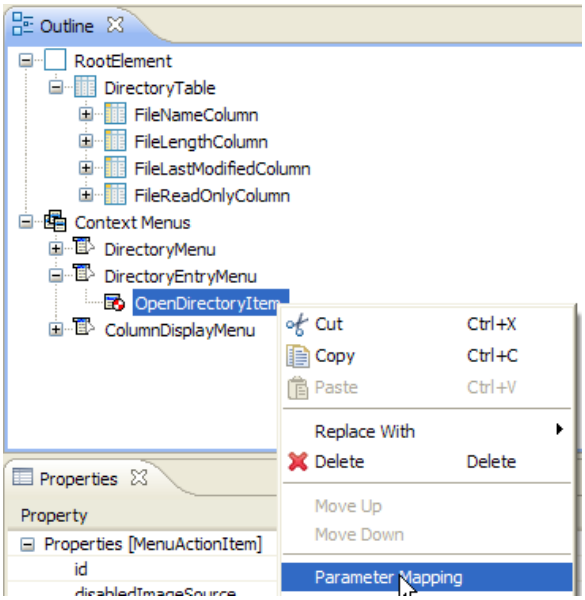


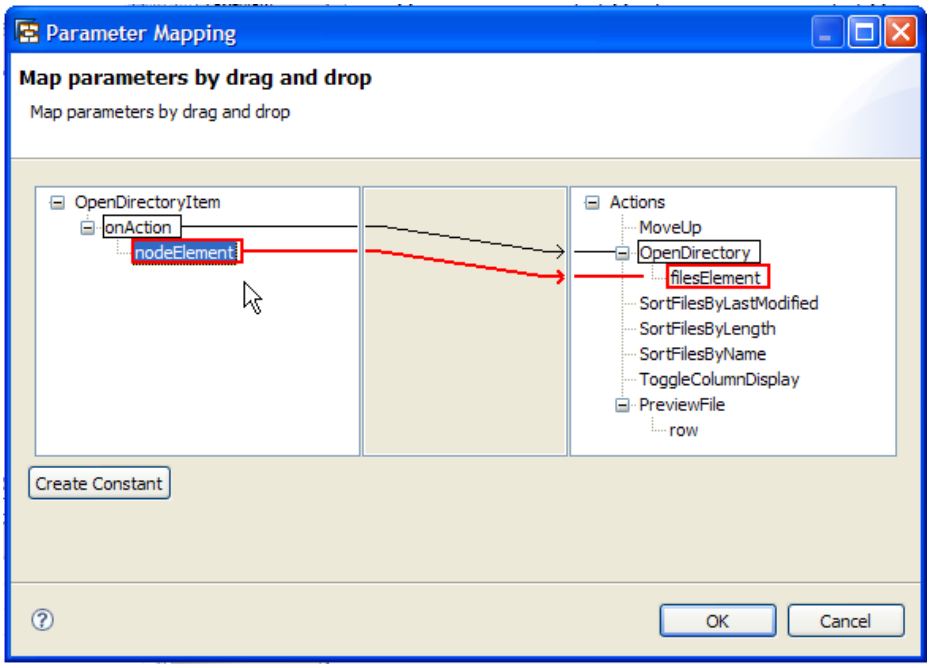
Let's have a look at the action for opening a directory. This action has to "know" from which table row it was triggered (through the context menu above that row). Therefore, we add a parameter `filesElement` of type `IFilesElement` to the action (`IFilesElement` is the runtime type of the elements of the table's data source node `Files`):



If you have mapped the context node from the component controller, make sure that you select the `IFilesElement` type from the view controller here.

For the `OpenDirectoryItem` menu item we define an event parameter mapping from the (implicit) `nodeElement` event parameter to the `filesElement` action parameter:



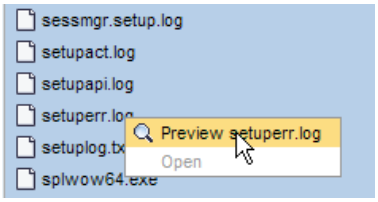


This parameter mapping has the effect that the action parameter "`filesElement`" will always contain the context element that represents the table row above which the context menu has been opened. (You cannot use the lead selection of the table because opening a context menu does not change the lead selection.)

Comment [D2]: It is important. May bei t makes sense to stress this point?

Modifying Context Menus at Runtime

For (some types of) text files, an additional context menu for previewing the file appears:



You achieve this by adding the following code to the `wdOnContextMenu()` view controller method that is called just before the context menu is opened:

```
  public final void wdOnContextMenu
  (
    final com.sap.tc.webdynpro.clientserver.uielib.standard.api.IWDContextMenuManager
contextMenuManager,
    final com.sap.tc.webdynpro.progmodel.api.IWDContextMenuEvent event
  )
  {
    //@@begin wdOnContextMenu
    IWDMenu dirEntryMenu = contextMenuManager.getContextMenu("DirectoryEntryMenu");
    if (contextMenuManager.getCurrentContextMenu() == dirEntryMenu)
    {
      modifyDirEntryMenu(dirEntryMenu, contextMenuManager, (IFilesElement)
event.getNodeElement());
    }
    //@@end
  }
```

The code first gets a runtime reference to the context menu “DirectoryEntryMenu” that has been defined at design time. It then checks whether the current context menu request would open this context menu. If this is the case, a method for modifying the context menu is called:

```
  private void modifyDirEntryMenu(IWDMenu contextMenu, IWDContextMenuManager
contextMenuManager, IFilesElement e)
  {
    IWDMenuActionItem item = null;
    if ("PreviewItem".equals(contextMenu.getItem(0).getId()))
    {
      item = (IWDMenuActionItem) contextMenu.getItem(0); // preview item already
exists
    }
    if (canPreview(e.getFile()))
    {
      if (item == null)
      {
        item = contextMenuManager.createActionItem("PreviewItem");
        item.setOnAction(wdThis.wdGetPreviewFileAction());

item.mappingOfOnAction().addSourceMapping(IWDMenuActionItem.IWDOnAction.NODE_ELEMENT,
"row");
        contextMenu.addItem(item, 0);
      }
      item.setVisible(true);
```

```
        if (e.getFile().length() > MAX_LENGTH_FOR_PREVIEW)
        {
          item.setEnabled(false);
          item.setText(tr(IMessageTutorial.CM_FILE_PREVIEW_DISABLED, e.getName()));
        }
        else
        {
          item.setEnabled(true);
          item.setText(tr(IMessageTutorial.CM_FILE_PREVIEW, e.getName()));
        }
      }
      else
      {
        if (item != null)
        {
          item.setVisible(false);
        }
      }
    }
```

This method first checks whether the “Preview” menu entry has already been created and added to the context menu. If the file that belongs to the table entry where the context menu has been opened can be previewed, the menu entry is made visible (and created if it does not yet exist). If the file is too large to be previewed, the menu entry is disabled and its text is changed.

Of course, it would have been simpler to just define the “Preview” entry already at design time and change its properties using data binding, but we want to illustrate that you still have full control over the context menu before it is opened.

You can also create or delete context menus from within this hook method using the `IWDContextMenuManager` API.. This API also allows you to change the assignment of context menus dynamically (see method `setCurrentContextMenu()`). This could be used to switch between a set of context menus defined at design time (depending on some condition known only at runtime) or to suppress a context menu at runtime (by setting the current context menu to `null`).

Finally let us have a look at the parameters of the predefined hook method `wdOnContextMenu()`:

```
    public final void wdOnContextMenu
    (
      final com.sap.tc.webdynpro.clientserver.uielib.standard.api.IWDContextMenuManager
  contextMenuManager,
      final com.sap.tc.webdynpro.progmodel.api.IWDContextMenuEvent event
    )
```

The `IWDContextMenuManager` interface provides methods for

- Creating all different menu items described above
- Creating and deleting (?) context menus, also those defined at design time
- Accessing any existing context menu by its ID
- Accessing and changing the “current” context menu. This menu is computed by the framework from the context menu assignments.

The `IWDContextMenuEvent` interface contains information about the current context menu request. The event stores:

- The *originator* of the request, that is the UI element where the right-click or keyboard shortcut (for example, Shift-F10) for opening the menu occurred.
- The *provider* of the context menu that will be opened. This is the UI element that provides the context menu that will be opened.

- The context element that represents the item above which the context menu has been opened. This is useful for all data-driven UI elements like tables and trees where the rows and nodes are defined by the context elements of their data source node.

Tutorial Result

We have demonstrated how to define and assign context menus at design time and how to access a context menu at runtime in the `wdOnContextMenu()` view controller method that is called just before the context menu is opened. You have learned how to modify a context menu at runtime depending on certain conditions about the object for which it has been opened.

The Web Dynpro programming model provides a declarative method for defining context menus and assigning them to screen areas together with an API for creating, modifying, and assigning context menus at runtime.

Text Symbols

Symbol	Usage
	Note
	Recommendation
	Warning
	See also

Copyright