

Creating a Web Dynpro Tree

Applies to:

Web Dynpro Java in SAP NetWeaver 7.0 (2004s)

Summary

This example demonstrates how to create, implement, deploy and run a basic Web Dynpro application using the [Tree](#) UI element to display an example file system in recursive manner. The tutorial also shows you how to implement a more performant approach in handling trees by loading the required data only.

Details

Level of complexity: Intermediate

Time required for completion: 60 min.

Sample Project

A ready to use project is available at [WDJ TUT Tree 70.zip](#) if desired.

Author: SAP NetWeaver Product Management

Company: SAP AG

Created on: 10 January 2008

Table of Contents

Task	3
Objectives	4
Creating a new Web Dynpro Project	4
Creating the Context for the TreeView	4
Creating the context	4
Creating Actions for the Tree	5
Creating UI Elements	7
Creating a Resource Bundle for the File Structure	8
Initialising the Context	10
Mapping the Event Parameters	12
Event Handling: Expanding a Node	13
Event Handling: Selecting an Entry	13
Building, Deploying, and Running the Project	13
Copyright	15

Task

This tutorial shows you how to use a performance-efficient recursive Web Dynpro tree. For that purpose, you will insert the Tree UI element in a view and implement its controller methods in such a way, that only when the user expands a node, the necessary data is added dynamically.

A Web Dynpro tree represents the exact structure as it exists in the context of the view at runtime. For instance, since it is impossible to know how many subfolders a recursive file system of an operating system has, the context must be constructed recursively as well. This can be achieved using a recursive context node.

In addition, if the user selects any file entry in the tree structure, the filename will be displayed in an InputField UI element.

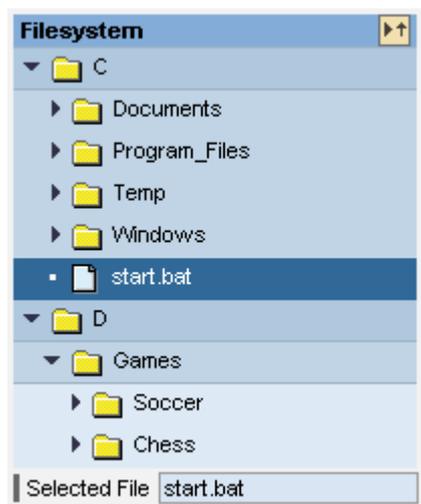
The view consists of a tree-interface element, a label, and an input field for displaying the selected element.

The development process is divided into a declaration part and an implementation part. In the declaration part, you execute the following steps:

1. Create the context for the *TreeView*
2. Create actions for the tree
3. Create the necessary UI elements, including their binding to the context

In the implementation part, you execute the following steps:

4. Create a resource bundle for the file structure
5. Initialize the context
6. Execute the parameter mapping of the UI element event parameter.
7. Event handling for expansion of a node



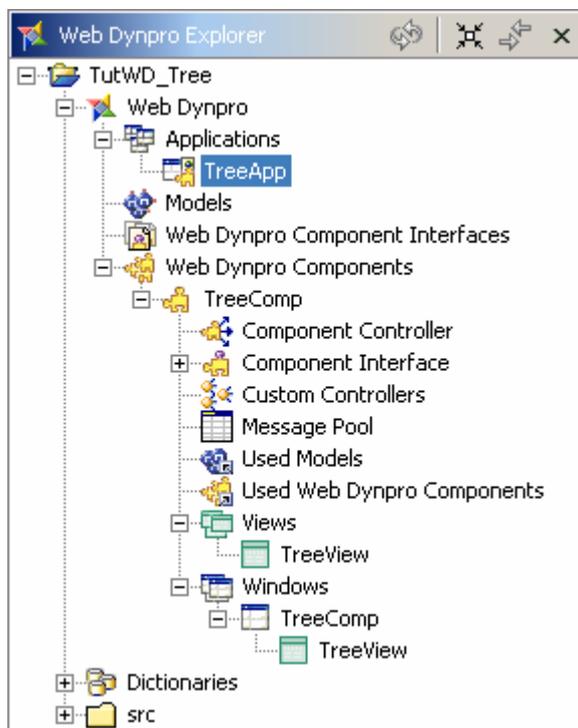
Objectives

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

- ✓ Use a context recursively
- ✓ Implement a Web Dynpro tree that loads data dynamically
- ✓ Implement an event handler that can react to the selection of a node of the tree.

Creating a new Web Dynpro Project

Create a new Web Dynpro Project with all its necessary units. Name the project **TutWD_Tree**, the application **TreeApp**, the component **TreeComp** and the view **TreeView**. Once you have created the project, you will see the following structure in the Web Dynpro Explorer:



Creating the Context for the TreeView

To display a recursive structure using the Tree UI element, the following context structure must be defined at design time. A *FolderContent* node is created containing some attributes and a *recursive node*. Some attributes can be bound to the *TreeNodeType* properties to control its behavior. For example, the *HasChildren* attribute is used to determine whether the current node is a leaf or not.

Creating the context

1. To open the *View Designer*, double-click *TreeView* in the project structure.
2. Choose the *Context* tab page.

3. Create the nodes and attributes displayed in the table below and set the properties.

Context Element	Type	Properties	Value
 FolderContent	Value node		
 ChildNode	Recursion Node	<u>repeatedNode</u>	TreeView.FolderContent
 HasChildren	Value attribute	type	<u>boolean</u>
 IconSource	Value attribute	type	<u>string</u>
 IgnoreAction	Value attribute	type	<u>boolean</u>
 IsExpanded	Value attribute	type	<u>boolean</u>
 Text	Value attribute	type	<u>string</u>
 TextOfSelectedNode	Value attribute	type	<u>string</u>

4. To save the project metadata, choose  Save all metadata from the toolbar.

Description:

- *FolderContent* represents a folder or a file in the tree
- *Text* stores the name of the item (folder or file)
- *HasChildren* specifies whether the node has children or not
- *IconSource* specifies the path to an icon
- *IsExpanded* expands a node initially
- *IgnoreAction* is used to override the execution of the onAction method.

Creating Actions for the Tree

If a file is selected, we want the name of the file to be displayed in another UI element, e.g. an *input field*. Nothing happens when a folder is selected, since the *IgnoreAction* attribute is set to *false*. And for a high-performance solution, we need an action bound to the *onLoadChildren* event, which loads the required data once.

1. To open the *View Designer*, double-click *TreeView* in the project structure.
2. Choose the *Actions* tab page.
3. Choose *New* to create a new action with the name **LoadChildren** and choose *Next*.
4. To add a new parameter, choose *New*. Give this parameter the name **element** and for *Type*, choose .
5. In the dialog box that appears, select *Java Native Type*, then choose *Browse...*. In the next dialog box that appears, enter **IFolderContentElement**.

6. Choose *OK* twice and *Finish* twice.
7. Repeat steps 3 to 6 for another action with the name **select** and the parameter **selectedElement** of the same type as in *LoadChildren*.

Creating UI Elements

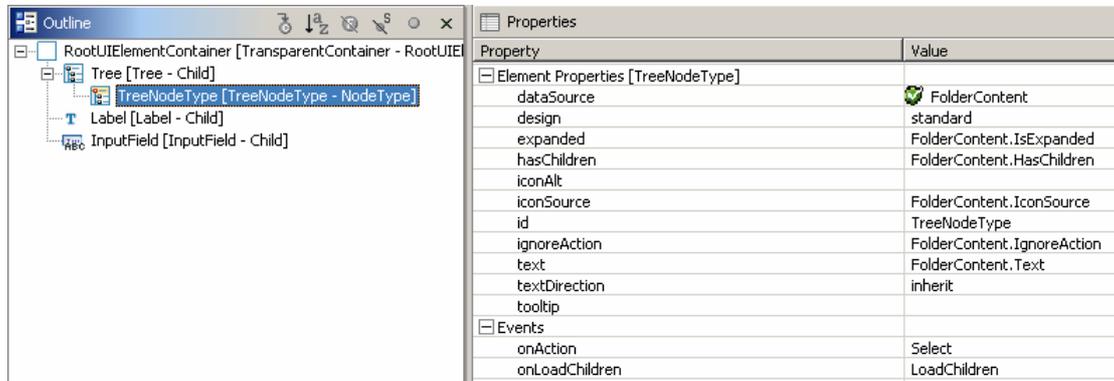
We want the view of this application to display a tree and the name of a selected file. In this step, you integrate a tree into the view and bind it to the corresponding context attributes and actions. You also create a label and an input field.

1. To open the *TreeView*, double-click *TreeView* in the project structure in the Web Dynpro Explorer.
2. Choose the *Layout* tab page and delete the *DefaultTextViewItem* in the Outline View.
3. Insert the UI elements as listed in the table below in the *Outline View* and set the properties in the *Properties* tab.

Property	Value
TheTree of type Tree	
<i>Properties of Tree – dataSource</i>	 FolderContent
<i>Properties of Tree – rootVisible</i>	false
<i>Properties of Tree – title</i>	Filesystem
<i>Properties of Tree – width</i>	200px
TheNode of type TreeNodeType as child of <i>TheTree</i>	
<i>Properties of TreeNodeType – dataSource</i>	 FolderContent
<i>Properties of TreeNodeType – expanded</i>	FolderContent.IsExpanded
<i>Properties of TreeNodeType – hasChildren</i>	FolderContent.HasChildren
<i>Properties of TreeNodeType – iconSource</i>	FolderContent.IconSource
<i>Properties of TreeNodeType – ignoreAction</i>	FolderContent.IgnoreAction
<i>Properties of TreeNodeType – text</i>	FolderContent.Text
<i>Event – onAction</i>	Select
<i>Event – onLoadChildren</i>	LoadChildren
Label of type Label	
<i>Properties of Label – labelFor</i>	Input  You cannot set this property until you have created the <i>Input</i> input field.
<i>Properties of Label – text</i>	Selected File
Input of type InputField	

<i>Properties of InputField – readOnly</i>	True
<i>Properties of InputField – value</i>	 TextOfSelectedNode

The *Outline View* and the *TreeNodeType* properties should look like this:



Creating a Resource Bundle for the File Structure

To be able to display some recursive content in a tree, we first need to simulate a file system. In this tutorial we will make use of a simple textfile stored in our project structure. To do so, we just create a resource file which contains all the necessary information. On startup, an initialize method then parses this file building up the dynamic context.

Add a new directory **resources** to the `src/packages/` directory.

1. Switch to the *Package Explorer* and choose the nodes *Tree > src/packages*
2. In the context menu, choose *New > Other*.
3. In the dialog box that appears, choose *Java* (left side) –  *Package* (right side).
4. Choose *Next*. In the window that appears, enter the *Package* name **resources** and confirm with *Finish*.

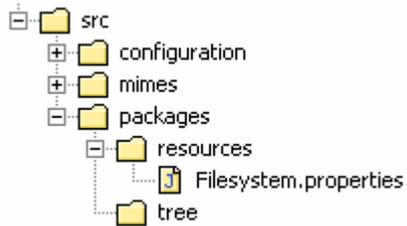
Save the file `Filesystem.properties` in the directory that you just created.

1. Select the node *Tree > src/packages/resources* and, in the context menu, choose *New > Other*.
2. In the dialog box that appears, choose *Simple* (left side) –  *File* (right side).
3. Choose *Next*. In the window that appears, enter the file name **Filesystem.properties** and choose *Finish*.
4. Copy the following lines of code in the text file and save it.

```
Drives = C;D
C = Documents;Program_Files;Temp;Windows;start.bat
Documents = Word;calc.xls;db.mdb
Word = first.doc;second.doc
Program_Files = Winzip;Accessories
Winzip = wz.com;wzinst.hlp;test.zip
Accessories = calculator.exe;notepad.exe;paint.exe
Windows = Java;Temporary_Internet_Files
```

```
Java = app.java;app.class
Temporary_Internet_Files =
cookie1.txt;cookie2.txt;cookie3.txt;cookie4.txt
D = Games
Games = Soccer;Chess
Soccer = soccer.exe;field.lnd
Chess = chess.exe
```

The project structure under the `src` folder:



Initialising the Context

Now we implement the reusable method `addChildren(parentNode)`, which adds a new node to the given parent node. The hook method `wdInit()` builds up the recursive context structure parsing the `Filesystem.properties` file.

1. To open the `TreeView`, double-click `TreeView` in the project structure in the Web Dynpro Explorer.
2. Choose the `Implementation` tab.
3. At the end of the file, after the `//@@begin others` tag, enter the following lines:

```
//@@begin others

// store resource handler for property resource file in private
// member variable
private IWDResourceHandler resourceHandlerForTree = null;

/**
 * Adds child elements to given parent element. The parent
 * element corresponds to a folder. The folder content (folder
 * elements) is stored as a list of context elements of type
 * IFolderContentElement within the non-singleton node 'ChildNode'
 * under the parent node.
 */
private void addChildren(
    IPrivateTreeView.IFolderContentElement parent) {
    IPrivateTreeView.IFolderContentNode folderContentNode =
        parent.nodeChildNode();
    IPrivateTreeView.IFolderContentElement folderContentElement;

    // read entries (folder content) for given key (folder)
    // in resource bundle
    String entries = resourceHandlerForTree.getString(parent.getText());
    StringTokenizer strTokenizer = new StringTokenizer(entries, ";");
    String anEntryToken;

    // if there is no entry in Filesystem.properties for given
    // key (parent.getText()) then the key is returned.
    if (entries.equals(parent.getText())) {
        folderContentElement =
            folderContentNode.createFolderContentElement();
        folderContentElement.setText("Empty Folder");
        folderContentElement.setHasChildren(false);
        folderContentElement.setIgnoreAction(true);
        folderContentNode.addElement(folderContentElement);
    } else {
        // populate non-singleton child node with node elements
        // (folder content elements)
        while (strTokenizer.hasMoreTokens()) {
            anEntryToken = strTokenizer.nextToken();
            folderContentElement =
                folderContentNode.createFolderContentElement();

            if (anEntryToken.indexOf(".") != -1) {
                // entryToken is a file
                folderContentElement.setHasChildren(false);
                folderContentElement.setIgnoreAction(false);
                folderContentElement.setIconSource("~/sapicons/s_b_crea.gif");
            } else {
                // entry token is a folder
                folderContentElement.setHasChildren(true);
                folderContentElement.setIgnoreAction(true);
                folderContentElement.setIconSource("~/sapicons/s_clofol.gif");
            }
        }
    }
}
```

```

        folderContentElement.setText(anEntryToken);
        folderContentElement.setIsExpanded(false);
        folderContentNode.addElement(folderContentElement);
    }
}
}
//@@end

```

4. Organize Imports: Press **Strg+Shift+o** or focus the source code area and select with the secondary mouse button **Source > Organize Imports** from the context menu.
5. Copy the following lines in the method `wdDoInit()` to obtain the initial tree structure:

```

public void wdDoInit()
{ //@@begin wdDoInit()
  //=== STEP 1: Load all initial Data from Filesystem.properties ===
  resourceHandlerForTree =
    WDResourceHandler.createResourceHandlerForCurrentSession();

  // Load the resource bundle "Filesystem.properties" located
  // in the package "com.sap.tut.wd.tree.resources"
  // for locale set in the resource handler.
  resourceHandlerForTree.loadResourceBundle(
    "resources.Filesystem",
    this.getClass().getClassLoader());

  // get all Drives
  String drives = resourceHandlerForTree.getString("Drives");
  StringTokenizer strTokenizer = new StringTokenizer(drives, ";");

  //=== STEP 2: Create node elements of the typ IFolderContentNode ==
  IPrivateTreeView.IFolderContentNode rootFolderContentNode =
    wdContext.nodeFolderContent();
  IPrivateTreeView.IFolderContentElement rootFolderContentElement;

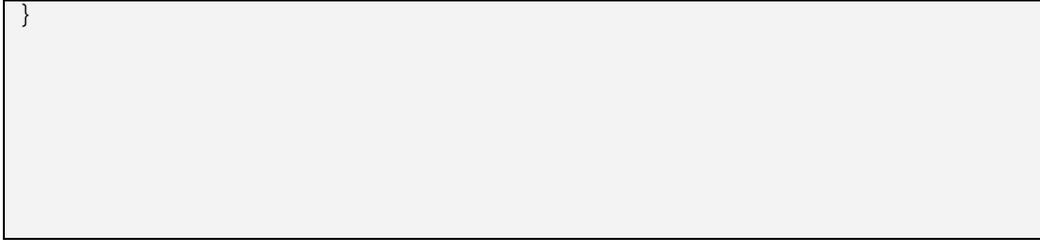
  // begin populating context node 'FolderContent'
  String aDriveToken;
  while (strTokenizer.hasMoreTokens()) {
    aDriveToken = strTokenizer.nextToken();

    // instantiate the new context node element of type
    // 'IFolderContentElement'
    rootFolderContentElement =
      rootFolderContentNode.createFolderContentElement();

    //set contained context value attributes
    rootFolderContentElement.setText(aDriveToken);
    rootFolderContentElement.setHasChildren(true);
    rootFolderContentElement.setIconSource(
      "~sapicons/s_clofol.gif");
    rootFolderContentElement.setIgnoreAction(true);
    // add root folder element to root folder node
    rootFolderContentNode.addElement(rootFolderContentElement);

    // if last folder node element, fill its non-singleton
    // child node (storing its entries) with children (folder node
    // elements) and expand the node ('Drive D' in tree).
    if (!strTokenizer.hasMoreTokens()) {
      addChildren(rootFolderContentElement);
      rootFolderContentElement.setIsExpanded(true);
    } else {
      rootFolderContentElement.setIsExpanded(false);
    }
  }
}
//@@end

```



6. Organize Imports (**Strg+Shift+o**) and **Save** the project.

Mapping the Event Parameters

In the second step, we created actions for events on a tree node and assigned each of them a parameter. To ensure that these parameters correspond to the selected node at runtime, you must implement a parameter mapping.

1. To open the *TreeView*, double-click *TreeView* in the project structure in the Web Dynpro Explorer.
2. Choose the *Implementation* tab page.
3. Enter the following lines of code in the method `wdDoModifyView()`:

```
public static void wdDoModifyView(IPrivateTreeView wdThis,
IPrivateTreeView.IContextNode wdContext,
com.sap.tc.webdynpro.progmodel.api.IWDView view, boolean
firstTime)
{
    /**@begin wdDoModifyView
        if (firstTime) {
            IWDTreeNodeType treeNode =
                (IWDTreeNodeType)
                view.getElement("TreeNodeType");

            /* parameter mapping from parameter "path" to
             * parameter "selectedElement"
             */
            treeNode.mappingOfOnAction().addSourceMapping(
                "path",
                "selectedElement");
            /* parameter mapping from parameter "path" to
             * parameter "element".
             */

            treeNode.mappingOfOnLoadChildren().addSourceMapping(
                "path",
                "element");
        }
    /**@end
}
```

4. Organize Imports (**Strg+Shift+o**) and **Save** the project.

These lines create a parameter mapping from the UI element parameter *path* to the event parameter *selectedElement* or *element*. The parameter *path* is of type *string* and contains the string representation of the tree node that triggers the event *onAction* or *onLoadChildren*.

Event Handling: Expanding a Node

The defined *LoadChildren* action and its parameter is used to load the required data once, since the browser will cache the data. The previously defined method *addChildren* can be used to take care of this.

1. To open the *TreeView*, double-click *TreeView* in the project structure in the Web Dynpro Explorer.
2. Choose the *Implementation* tab page.
3. Enter the following line in the method *onActionLoadChildren()* between the `//@@begin` and `//@@end` tags:

```
addChildren(element);
```

Event Handling: Selecting an Entry

We have defined the action *Select*, which contains the selected node as a parameter. Due to the parameter mapping, this parameter contains all the information we need at runtime about the current node. In this scenario, we only make use of the node's *getText()* accessor method to retrieve the name of the item and set the context attribute *TextOfSelectedNode* accordingly which is bound to our input field. And since the *IgnoreAction* is set to *true* for nodes representing folders, the method won't be executed.

1. To open the *TreeView*, double-click **TreeView** in the project structure in the Web Dynpro Explorer.
2. Choose the *Implementation* tab page.
3. Enter the following lines in the method *onActionSelect()* between the `//@@begin` and `//@@end` tags:

```
wdContext.currentContextElement().setTextOfSelectedNode(
    selectedElement.getText());
```

Building, Deploying, and Running the Project

An example of how to deploy and run your application you can find [here](#).

Related Content

[Recursive Context Nodes](#)

Copyright

© Copyright 2007 SAP AG. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.

Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.

Microsoft, Windows, Outlook, and PowerPoint are registered trademarks of Microsoft Corporation.

IBM, DB2, DB2 Universal Database, OS/2, Parallel Sysplex, MVS/ESA, AIX, S/390, AS/400, OS/390, OS/400, iSeries, pSeries, xSeries, zSeries, System i, System i5, System p, System p5, System x, System z, System z9, z/OS, AFP, Intelligent Miner, WebSphere, Netfinity, Tivoli, Informix, i5/OS, POWER, POWER5, POWER5+, OpenPower and PowerPC are trademarks or registered trademarks of IBM Corporation.

Adobe, the Adobe logo, Acrobat, PostScript, and Reader are either trademarks or registered trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Oracle is a registered trademark of Oracle Corporation.

UNIX, X/Open, OSF/1, and Motif are registered trademarks of the Open Group.

Citrix, ICA, Program Neighborhood, MetaFrame, WinFrame, VideoFrame, and MultiWin are trademarks or registered trademarks of Citrix Systems, Inc.

HTML, XML, XHTML and W3C are trademarks or registered trademarks of W3C®, World Wide Web Consortium, Massachusetts Institute of Technology.

Java is a registered trademark of Sun Microsystems, Inc.

JavaScript is a registered trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.

MaxDB is a trademark of MySQL AB, Sweden.

SAP, R/3, mySAP, mySAP.com, xApps, xApp, SAP NetWeaver, and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world. All other product and service names mentioned are the trademarks of their respective companies. Data contained in this document serves informational purposes only. National product specifications may vary.

These materials are subject to change without notice. These materials are provided by SAP AG and its affiliated companies ("SAP Group") for informational purposes only, without representation or warranty of any kind, and SAP Group shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP Group products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

These materials are provided "as is" without a warranty of any kind, either express or implied, including but not limited to, the implied warranties of merchantability, fitness for a particular purpose, or non-infringement.

SAP shall not be liable for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of these materials.

SAP does not warrant the accuracy or completeness of the information, text, graphics, links or other items contained within these materials. SAP has no control over the information that you may access through the use of hot links contained in these materials and does not endorse your use of third party web pages nor provide any warranty whatsoever relating to third party web pages.

Any software coding and/or code lines/strings ("Code") included in this documentation are only examples and are not intended to be used in a productive system environment. The Code is only intended better explain and visualize the syntax and phrasing rules of certain coding. SAP does not warrant the correctness and completeness of the Code given herein, and SAP shall not be liable for errors or damages caused by the usage of the Code, except if such damages were caused by SAP intentionally or grossly negligent.