# Constructing a Recursive and Loadable Web Dynpro Tree

SAP NetWeaver 04

## Copyright

## Icons in Body Text

| Icon | Meaning |
|---|---|
| ⚠ | Caution |
| ▣ | Example |
| 💡 | Note |
| 🧭 | Recommendation |
| syn | 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. |
| | 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`. |

Constructing a Recursive and Loadable Web Dynpro Tree ..... 5
Importing a Project Template ..... 6
Recursive Context Nodes ..... 7
Further Procedure for Creating the Web Dynpro Tree ..... 8
Creating the Context for the TreeView ..... 9
Creating Actions for the Tree ..... 10
Creating UI Elements ..... 11
Creating a Resource Bundle for the File Structure ..... 12
Initialising the Context ..... 14
Mapping the Event Parameters ..... 16
Event Handling: Expanding a Node ..... 17
Event Handling: Selecting an Entry ..... 18
Building, Deploying, and Running the Project ..... 19

# Constructing a Recursive and Loadable Web Dynpro Tree

In the following tutorial, you construct a Web Dynpro tree that only loads its data when the user expands a node of the tree.

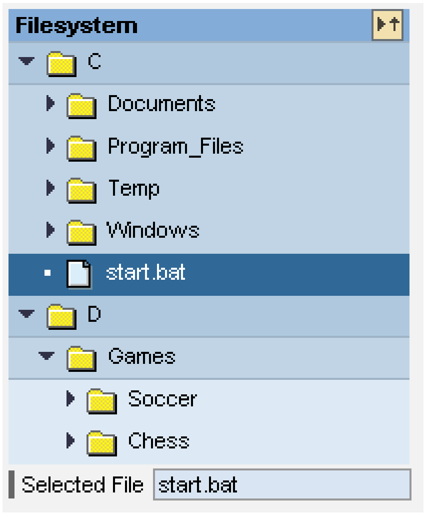
## The Task

This tutorial shows you how to use a performance-efficient recursive Web Dynpro tree. For that purpose, you will insert the tree interface-element in a view. Furthermore, you will implement the controller of the view in such a way, that only when the user expands a node, the direct sub-nodes and sub-elements of the node are added.

In this tutorial, you simulate a file structure that is to be displayed in the form of a tree. A Web Dynpro tree represents the exact structure as it exists in the context of the view at runtime. Since it is impossible to know how many subfolders the file structure has, the context must be constructed recursively. The recursive structure has the advantage that any number of subfolders can exist at runtime.

You also implement the selection of an entry from the tree-interface element. In this tutorial, you want to display the text of the selected entry in an input field.

You want the user interface of this Web application to have only one view. This view consists of a tree-interface element, a label, and an input field for displaying the selected element. Initially, you want a node of the tree-interface element to be selected and expanded.



## 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
- ✓ Implement an event handler that can react to the selection of a node of the tree.

## Prerequisites

### Systems, Installed Applications, and Authorizations

- SAP NetWeaver Developer Studio is installed on your PC.
- You have access to the SAP J2EE Engine.

### Knowledge

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

## Next step:

Importing a Project Template [page 6]

# Importing a Project Template

On the SAP Developer Network (SDN) at http://sdn.sap.com, the following Web Dynpro projects are available for this tutorial:

- The project template *TutWD_Tree_Init* (the starting point for this tutorial)
- The completed Web Dynpro project *TutWD_Tree* (corresponds to the project *TutWD_Tree_Init* after completion of the tutorial)

The projects are available for download in corresponding zip files under the category *Home* → *Developer Areas* → *Web Application Server* → *Web Dynpro*.

## 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 to the SAP NetWeaver Developer Studio

1. Call the SAP NetWeaver Developer Network by using the URL http://sdn.sap.com and log on with the corresponding user ID and password. If you do not have a user ID, you must register before you can proceed.
2. Download the zip file *TutWD_Tree_Init.zip* containing the project template *TutWD_Tree_Init* and save the zip file to any directory on your local hard disk or directly in the work area of the SAP NetWeaver Developer Studio.
3. Extract the content of the zip file *TutWD_Tree_Init.zip* in the work area of the SAP NetWeaver Developer Studio or in any directory on your local hard disk.
4. Call the SAP NetWeaver Developer Studio.
5. Import the Web Dynpro project *TutWD_Tree_Init*.
   a. In the menu, choose *File* → *Import*.

b. In the next window, choose *Existing Project into Workspace* and choose *Next* to confirm.

c. Choose *Browse*, open the folder in which you saved the project *TutWD_Tree_Init*, and select the project.

d. Choose *Finish* to confirm.

6. The Web Dynpro project *TutWD_Tree_Init* appears in the Web Dynpro Explorer for further processing and completion of the tutorial.

## Initial Project Structure

Once the Web Dynpro project template *TutWD_Tree_Init* has been imported, the following project structure is displayed in the Web Dynpro Explorer:

| **Web Dynpro Project Structure** |
|---|
| Web Dynpro project: **`TutWD_Tree_Init`** |
| Web Dynpro application: **`TreeApp`**<br>The application *TreeApp* displays the interface view of the Web Dynpro component *TreeComp* in the browser window. |
| Web Dynpro component: **`TreeComp`**<br>Web Dynpro component containing our entire application. |
| View: **`TreeView`**<br>In this view, the tree is displayed and you can expand and collapse nodes and select entries. |
| Window: **`TreeComp`**<br>Contains the *TreeView* and displays it when the program is started. |

### Next step:

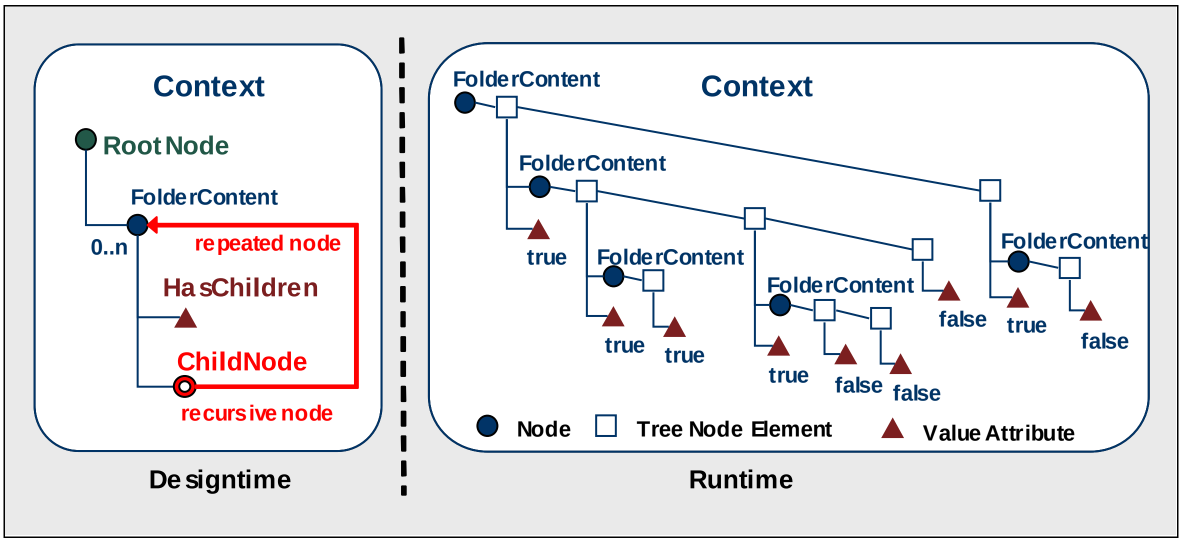
Recursive Context Nodes [page 7]

## Recursive Context Nodes

In some applications, it may be necessary to display recursive data in the context. A simple example for the use of recursive data is the file system of an operating system. It consists of folders and files, and each folder can contain further folders and files. When defining controller contexts, it should be possible to display such recursive data structures.

In Web Dynpro, this is implemented as follows. Under the parent node, you insert a special context node, a *recursive node*. You then assign this node the property *repeated node*. The value of this property corresponds to the higher-level context node that is to be *repeated*. At runtime, a recursive node is automatically of the non-singleton type and therefore exists exactly once for each parent node element.

To display a directory structure using the tree UI element, the following context structure is defined in the example: At design time, a *FolderContent* node is created that can represent both a folder and a document. Using the context attribute *HasChildren*, you can specify whether the node element represents a folder (`HasChildren: true`) or a file (`HasChildren: false`). Therefore, the context has the following structure at design time or runtime:



## Next step:

[Further Procedure for Creating the Web Dynpro Tree [page 8]]

## Further Procedure for Creating the Web Dynpro Tree

The individual steps for creating the tree UI-element are presented below.

### Process Flow

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
8. Event handling for selection of an entry

### Next step:

Creating the Context for the TreeView [page 9]

## Creating the Context for the TreeView

Before you create the context for the *TreeView*, you must decide which data you want to display in the tree. This includes visible data, for example the text of a node, and invisible data, for example, whether a node can be selected.

The objective of this example application is to display a file structure in a tree and display the name of the selected file. This means that you need an attribute in the context in which the currently selected name is stored and can be displayed. Several context attributes are required for the nodes of the tree. The application requirements are as follows:

- Each node must be labeled.
- A file must have a different icon to a folder
- A file is a leaf and can therefore have no further leaves or nodes
- One node must already be opened
- Selecting a file must trigger an event that displays the name of the file on the screen; selecting a folder must not trigger an event.

This results in several context attributes that you create in the TreeView context.

The TreeView is already contained in the template.

### Procedure

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 | `repeatedNode` | `TreeView.FolderContent` |
| HasChildren | Value attribute | `type` | `boolean` |
| IconSource | Value attribute | `type` | `string` |
| IgnoreAction | Value attribute | `type` | `boolean` |
| IsExpanded | Value attribute | `type` | `boolean` |
| Text | Value attribute | `type` | `string` |

| | | | |
|---|---|---|---|
| TextOfSelectedNode | Value attribute | `type` | `string` |

The tree UI-element represents a hierarchy as it is saved in the context. If the tree is not recursive, then the entire desired structure in the context must be displayed with the help of context-nodes and context-attributes. In doing so, bear in mind that the child nodes possess the *singleton*-property ***false***. Thereby it is achieved that there exist a child node instance for every node element in the parent node.

If the *singleton*-property was *true*, only one single node instance would be represented, whose content would change every time the lead-selection of the parent node changed. Thus, the context for the unselected node elements would not contain child nodes, so that in the tree-UI-element there would be no data displayed at the unselected places.

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

The *FolderContent* context element later displays a folder or a file in the tree of the *TreeView*. The name of the folder or the file is saved in the *Text* attribute. The *HasChildren* attribute specifies whether the context-node displays a node or a leaf. You can use this attribute to set the *IconSource* to a folder icon or file icon. The *IsExpanded* attribute is required to expand a node initially. You use the *IgnoreAction* attribute to define whether or not the event *onAction* is triggered when this node is selected.

The recursive *ChildNode* node links to the *FolderContent*, which can thus be nested as deeply as you like.

## Result

The *TreeView* context is defined and its file structure can be nested as deeply as is required. It also has the corresponding properties for displaying the tree as requested.

### Next Step:

Creating Actions for the Tree [page 10]

## Creating Actions for the Tree

Before you define actions, you must clarify which options will be available to the user of the application.

You want the user of this application to be able to expand the folders of the tree and select files. When a folder is expanded, you want all the files it contains to be displayed, as well as any subfolders. If a file is selected, you want the name of the file to be displayed on the screen. Nothing happens when a folder is selected.

There are two ways to implement the expansion of the folder. It is possible to load the whole file structure into the TreeView context during initialization of the *TreeView*. This variant is not suitable in this case, since the large file structure would mean a long loading process and transfer of a large amount of data. A more performance-efficient solution would be preferable.

- For this reason, the nodes of a tree UI-element have the event *onLoadChildren*, which is triggered whenever a node with no data is expanded. You can bind an action to this event to load all the necessary data belonging to the current node. In this tutorial, the data to be loaded comprises the folders and files located directly underneath the selected folder. A parameter value must be transferred to the action event handler, to enable it to know which node has been selected.

A tree node also has the event *onAction*. This is triggered whenever the user selects a tree entry. In this example, the user can select a file or a folder, but the event is only triggered when a file is selected. You have to define an appropriate action to be able to react to the selection of a file. The event of the tree-node can then be bound to this action. You must also declare a parameter for this action to save the selected element.

## Procedure

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*.

## Result

You have defined the necessary actions that you will bind to the corresponding events of the tree later. These actions enable you to react to user entries. The implementation of the methods is covered later in the tutorial.

### Next step:

Creating UI elements [page 11]

# Creating UI Elements

You 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.

## Procedure

1. To open the *TreeView*, double-click **TreeView** in the project structure in the Web Dynpro Explorer.
2. Choose the *Layout* tab page.
3. Create the UI elements displayed in the table below and set the properties.

| Property | Value |
|---|---|
| TheTree of type *Tree* | |

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

## Result

You have created a tree with a node of type *TreeNodeType* in the *TreeView* and bound it to the context attributes. You have also placed an input field in the view and bound it to a context attribute to enable the name of the selected file to be displayed later.

### Next step:

Creating a Resource Bundle for the File Structure [page 12]

# Creating a Resource Bundle for the File Structure

You want the structure of the file system simulated in the tree to be saved in a resource bundle. This resource bundle contains all specifications on the folders and files that exist in the file system.

## Procedure

Add a new directory **`resources`** in the `src/packages/com/sap/tut/wd/tree/` directory.

1. Switch to the *Package Explorer* and choose the nodes *Tut_WD_Tree_Init → src/packages → com.sap.tut.wd.tree.*
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 **`com.sap.tut.wd.tree.resources`**.
5. To complete this procedure, choose *Finish*.

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

1. Select the node *Tut_WD_Tree_Init → src/packages → com.sap.tut.wd.tree.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`**.
4. Choose *Finish*.
5. Enter the following lines in the text file and save the text file.

*`Filesystem.properties`*: Resource bundle for the file system

**(in the directory `src/packages/com/sap/tut/wd/tree/resources`)**

```
##############################################################
##
## Filesystem resources for Web Dynpro Tutorial Tree
##
##############################################################

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
```

## Result

You have created a directory called *resources* in the current project. You have created a resource bundle in this directory for the simulation of the file system with the name *Filesystem.properties*. You use this file in the next step to display a file structure in the tree.

**Next Step:**

Initializing the Context [page 14]

## Initialising the Context

In the previous steps, you have created the context structure that is to represent the file system. You have also created a resource bundle containing the file structure.

In this step, you will merely load the content of the resource bundle to the context.

To do this, you load the resource bundle in the method *wdDoInit()* and add the required folder to the context. You also set the individual context attributes, to which the individual tree nodes are bound. This includes the icons (so that a folder gets a folder icon and a file gets a file icon), the folders (so that they cannot be selected), the files that are leaves and not nodes, and the name of the folder or file to be displayed.

### Procedure

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 `//@@begin others`, 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
```

To add the missing imports, position the mouse pointer in the source code area, click the secondary mouse button, and choose *Source → Organize Imports*.

Since the resource bundle is accessed in the methods *wdDoInit()* and *addChildren(…)*, define a global variable called *resourceHandlerForTree* in the view controller. This saves you entering source code, because the connection to the file must only be established once.

The method *addChildren(…)* is created to save code lines, because it can be used in the method *wdDoInit()* and also later, when opening a folder, to add the child nodes to the current node.

4. Enter 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(
    "com.sap.tut.wd.tree.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
  }
```

To add the missing imports, position the mouse pointer in the source code area, click the secondary mouse button, and choose *Source* → *Organize Imports*.

## Result

You have defined a method that enables children to be added to a node. You have also saved data in the context of the method *wdDoInit()* that creates the appearance of the view as required in the user interface template at runtime.

### Next Step:

Mapping the Event Parameters [page 16]

# Mapping the Event Parameters

In the second step, you 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.

## Procedure

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 **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("TheNode");

         /* 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
  }
```

To add the missing imports, position the mouse pointer in the source code and in the context menu, choose `Source` → `Organize Imports`.

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 triggered the event *onAction* or *onLoadChildren*. You have already created the parameters *selectedElement* and *element* in the step Creating actions for the Tree [page 10].

## Result

You have executed a parameter mapping. You can use the parameters *element* and *selectedElement* in the event handlers to access the node selected by the user.

### Next step:

Event Handling: Expanding a Node [page 17]

## Event Handling: Expanding a Node

You want the user of this application to be able to expand nodes in the tree to display the folders and files that they contain.

As mentioned before, there are two opportunities to create the content of the tree that is to be displayed. In the less performance-efficient variant, you create the entire content of the tree during initialization and transfer a large amount of data to the browser. This would certainly be a viable option in the case of a small tree. In this case, no event handling, no *LoadChildren* action, and no parameter mapping is required. Since the context is filled in the *TreeView*, you can expand and collapse nodes without establishing a connection to the server.

However, in this application it is possible that a very large amount of data will be displayed in the tree and that the user will not expand every node. Therefore, it is more efficient to only load the data that the user wants to see into the context. You have already made the necessary steps to meet the requirements so that data is only loaded when a node is expanded for the first time. You have defined the *LoadChildren* action, which contains a node element as a parameter. You have bound this action to the event *onLoadChildren* of the tree node. You have also implemented a parameter mapping between the UI element event and the action event handler.

You now need to carry out the last step, which is to add the required source code in the method *onActionLoadChildren(…)*, to load data for the selected node. This method is only executed once for each node, because, as already described, the data is then available in the browser and does not have to be loaded again.

You can reuse the private method *addChildren(…)* that you created for context initialization purposes.

## Procedure

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 `onActionLoadChildren()`:

```
public void
onActionLoadChildren(com.sap.tc.webdynpro.progmodel.api.IWDCustomEvent
wdEvent, com.sap.tut.wd.tree.wdp.IPrivateTreeView
.IFolderContentElement element )
  {
    //@@begin onActionLoadChildren(ServerEvent)
      addChildren(element);
    //@@end
  }
```

## Result

You have added all the necessary lines in the method *onActionLoadChildren(..)* to load data for a node when it is selected.

### Next step:

Event Handling: Selecting an Entry [page 18]

# Event Handling: Selecting an Entry

When a file in the tree is selected, you want its name to be displayed in the input field. To make this possible, you have created the attribute *ignoreAction* in the context, to which the parameter *ignoreAction* of the tree node is bound. This parameter decides whether the event *onAction* is triggered or not when a node is selected. You have set this attribute so that it is always *true* when a folder is displayed and *false* when a file is displayed. (see step: Initialising the Context [page 14])

You have also defined the action *Select*, which contains the selected node as a parameter. To fill this parameter with the correct value at runtime, you have executed a parameter mapping.

In this step, you now transfer the name of the selected element to the *textOfSelectedNode* context attribute to display the file name in the input field.

## Procedure

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()`:

```
public void
onActionSelect(com.sap.tc.webdynpro.progmodel.api.IWDCustomEven
t wdEvent, com.sap.tut.wd.tree.wdp.IPrivateTreeView
.IFolderContentElement selectedElement )
  {
    //@@begin onActionSelect(ServerEvent)
    wdContext.currentContextElement().setTextOfSelectedNode(
         selectedElement.getText());
    //@@end
  }
```

## Result

You have added all the necessary lines in the method *onActionSelect(..)* to display the name of a selected file in the input field.

### Next step:

Building, Deploying, and Running the Project [page 19]

# Building, Deploying, and Running the Project

Once you have reached this stage, you can start and test the fully-developed example application in the Web Browser.

## Prerequisites

☐ You have made sure that the SAP J2EE Engine has been launched.

## Procedure

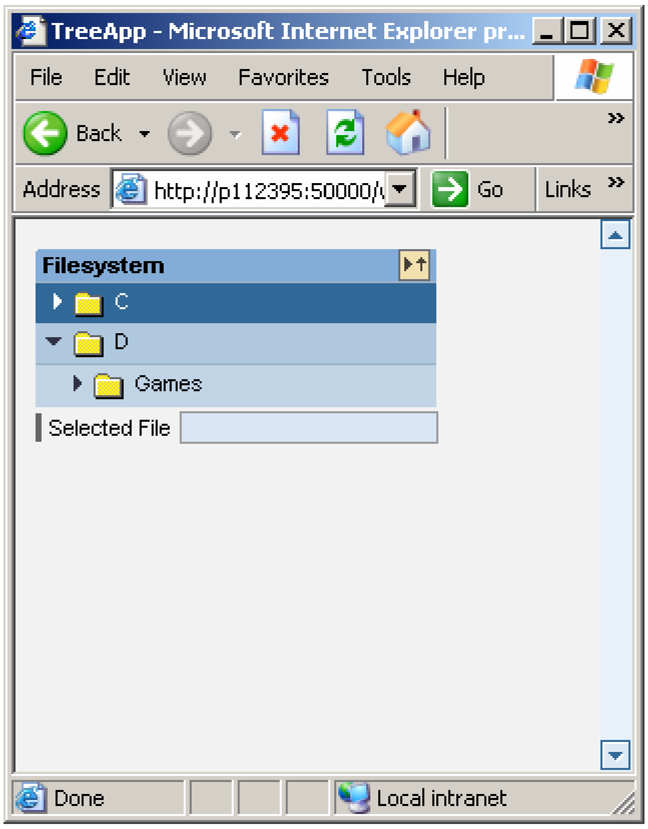
1. Save the current state of the metadata for your project by choosing *Save All Metadata*.
2. Open the context menu for the project node (**`TutWD_Tree_Init`**) in the Web Dynpro Explorer and choose *Rebuild Project.*

3. In the Web Dynpro Explorer, in the context menu of the application object **`TreeApp`**, choose *Deploy new archive and run*.

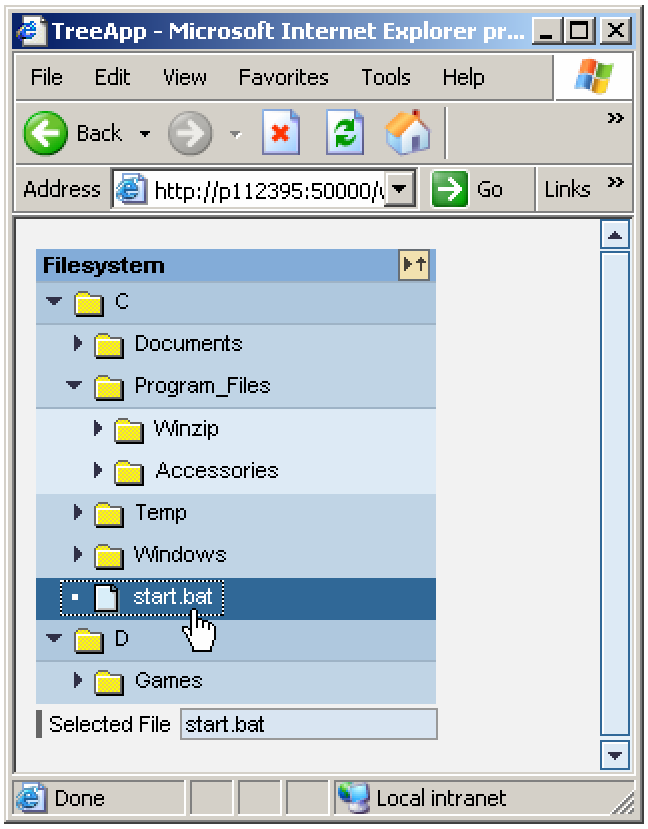
## Result

The Developer Studio performs the deployment process in one single step, based on an automatically generated Enterprise Archive File, and then automatically launches your application in the Web Browser.

The initial representation of the tree is displayed in the Web browser.



To test your Web application, expand and collapse individual nodes, and select files. Note that when you open a node a second time, no connection is established to the server since all the data has already been loaded in the browser.


TreeApp - Microsoft Internet Explorer pr...
File Edit View Favorites Tools Help
Back
Address http://p112395:50000/
Go
Links
Done
Local intranet

Filesystem
- C
  - Documents
  - Program_Files
    - Winzip
    - Accessories
  - Temp
  - Windows
  - start.bat
- D
  - Games

Selected File start.bat