



**How-to Guide
SAP NetWeaver '04
SAP Enterprise Portal 6.0**

How To... Build Dynamically Propogated Tree

Version 1.00 – July 2004

**Applicable Releases:
SAP NetWeaver '04
(• NW Support Package Stack 1)**

© Copyright 2004 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, z/OS, AFP, Intelligent Miner, WebSphere, Netfinity, Tivoli, and Informix are trademarks or registered trademarks of IBM Corporation 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.

SAP NetWeaver "How-to" Guides are intended to simplify the product implementation. While specific product features and procedures typically are explained in a practical business context, it is not implied that those features and procedures are the only approach in solving a specific business problem using SAP NetWeaver. Should you wish to receive additional information, clarification or support, please refer to SAP Consulting.

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.

1	Scenario.....	2
1.1	Overview.....	2
1.2	Overall WorkFlow	2
2	Prerequisites.....	3
3	Step-By-Step Solution	4
3.1	Creating the pool.....	4
3.2	Classes description.....	5
3.3	Tree renderer class	6
3.4	Calling the tree renderer class from the portal component.....	10
3.4.1	Sample Code	11
4	Risks.....	12
5	Appendix.....	13

1 Scenario

You are required to develop a portal component that display dynamically propagated tree with possibility to add nodes, assign icons to the nodes (i.e. directory and file names , database tables and their columns , XML file hierarchy and etc.), assign client side events . HTMLB provides basic implementation for representing hierarchical data but it is usually don't address most customer needs - it can't dynamically expand and collapse elements, assign icons to the nodes and etc.

1.1 Overview

The tree view is used to display hierarchical data or text. The hierarchy levels may be expanded and collapsed and every leaf of the tree is loaded only once . Every tree node contains a text and an image icon that expands and collapses the node or represents the tree node. It is possible to assign different icon to types of the node. The node text might also link to a function that displays the connected data. The tree view is using HTMLB styles - the first four levels have different colors. From the 5th level on the color stays the same like in the 4th level.

Usage of the tree renderer provides an easy way to create a tree by extending the abstract class **AbstractTreeRenderer** and providing data model objects – ITree (and ITreeNode) defined by UIService .

Tree Renderer is ideally suited to view Enterprise Business Objects such as database tables or SAP objects as a tree, and to enable selection of object properties (like database fields). Since loading of object children might be time consuming in these cases, it is required to load the children on-demand.

1.2 Overall WorkFlow

The DynamicTreeService of SAP Enterprise Portal is delivered in a portal archive (PAR file) named [com.sap.portal.productivity.util.dyntree.par](#) which contains all the Java and JavaScript functionality for manipulating the tree structure.

In a deployed portal, the PAR file is located at <SAPJ2EEEngine-deployment-dir>\cluster\server\services\servlet_jsp\work\jspTemp\irj\root\WEB-INF\deployment\pcd. During initial portal deployment, it is renamed to [com.sap.portal.productivity.util.dyntree.par.bak](#).

Functionality of tree depends on code of the following modules:

- UIService
- DynamicTreeService

To create a tree, we recommend the following procedure:

- Create Portal Component Object which will represent dynamically loading tree.
- Create Node Object that holds the node information such as – node id, node name, node type and etc.
- Create Class that extending **AbstractTreeRenderer** object and implement abstract functions.
- Add icons images to the portal component resources representing the tree nodes.

2 Prerequisites

System:

SAP Enterprise Portal 6.0 SP2 or higher
SAP J2EE Engine Version 6.2 or higher

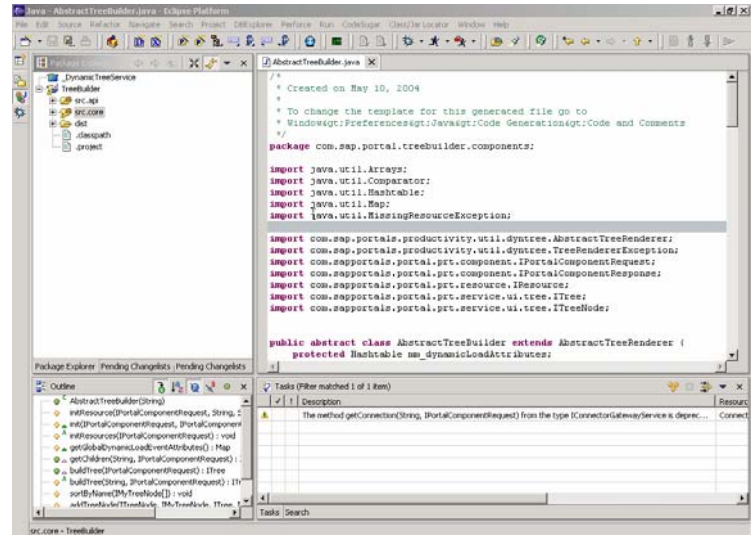
Developer:

The developer should have knowledge in EP 6.0 Portal Administration, Eclipse development environment with SAP Portal Plugin installed and at least medium level of java knowledge.

3 Step-By-Step Solution

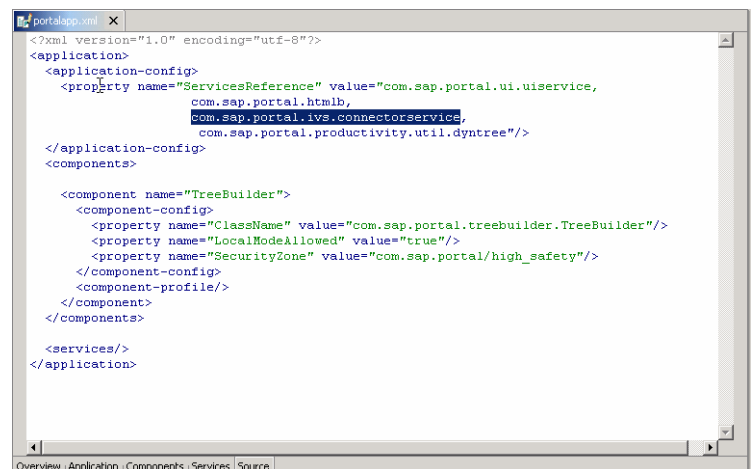
3.1 Creating the pool

1. Open Eclipse development Environment.
2. Create new Portal Application project or edit an existing one.
3. Create new Portal Component Object

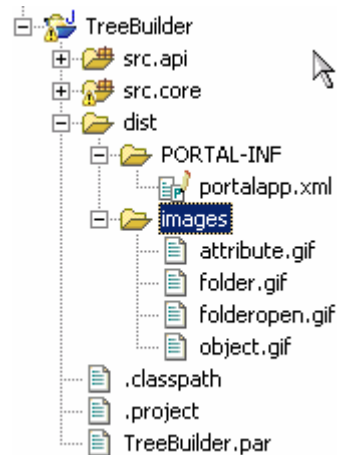


4. Edit portalapp.xml file located in dist\PORTAL_INF\portalapp.xml
5. Now you will have to add the reference to the relevant services:
 - [com.sap.portal.ui.uiservice](#)
 - [com.sap.portal.htmlb](#)
 - [com.sap.portal.productivity.util.dyntree](#)
 - [com.sap.portal.productivity.utils](#)

If your component will use additional PAR files please add them to the list. Additional reference to [com.sap.portal.ivs.connectorservice](#) was added in my example because of performing database connection.



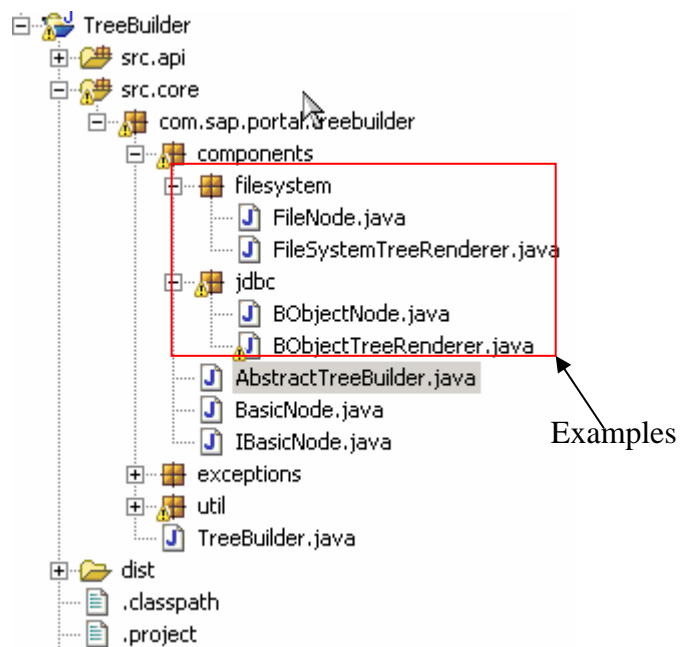
- If you want your tree to display icons please add the images to .../dist/PORTAL_INF/images directory.



6. Create two classes :

1) First class will represent the tree node. This class can be inherited from **BasicNode** class (which implement **IBasicNode** interface) provided in attached example. This class has the basic functionality such as node id, name, type, load status.

2) Second class should be inherited from **AbstractTreeRenderer** class or **AbstractTreeBuilder** class provided in attached example which provides additional functionality for sorting the nodes by name , adding the node to the tree , resource initialization and etc.



3.2 Classes description

- Tree Node Class

This is a class that implements the tree node definition. Every tree node should have the following properties:

- 1) **Node Id** – unique string which serve mostly for adding dynamically loaded nodes, retrieving the node from the tree. The node Id returned when you would like to know which node was clicked/doubleclicked, which node dropped/dragged and etc. Please see the ITree API from [com.sapportals.portal.prt.service.ui service](http://com.sapportals.portal.prt.service.ui.service).
- 2) **Node Name** – The node name displayed in the tree view.
- 3) **Node type** – The tree node can be of various types such as: folder, document , table, field and etc.

A **BasicNode** class supplied in attached example provides basic Node futures. You can either create your own class for representing the node or inherit from **BasicNode** provided in example and add additional functionality/properties in case **BasicNode** not completely meet your requirements..

The following example shows the basic node class:

```
package
com.sap.portal.treebuilder.components.hardcodedcontent;

import com.sap.portal.treebuilder.components.BasicNode;

public class MyNode extends BasicNode {

    public MyNode(String key, String name, boolean folder) {

        setId(key);

        setIsFolder(folder);
        if ((name != null) && (name.length() > 0)) {
            mm_name = name;
        } else {
            mm_name = "EMPTY";
        }
    }
}
```

3.3 Tree renderer class

This is the class that doing the real work and building the tree structure. In order to create this class you can either inherit from **AbstractTreeRenderer** class that belongs to [com.sap.portal.productivity.util.dyntree](#) service or inherit from **AbstractTreeBuilder** class provided in attached example.

Steps for creating the tree renderer class inherited from **AbstractTreeBuilder** ::

1. Create new class which inherit from **AbstractTreeBuilder** class:

```
import java.util.MissingResourceException;

import
com.sap.portal.treebuilder.components.AbstractTreeBuilder;
import com.sap.portal.treebuilder.components.IBasicNode;
import
com.sap.portals.productivity.util.dyntree.TreeRendererExcept
ion;
import
com.sapportals.portal.prt.component.IPortalComponentRequest;
import com.sapportals.portal.prt.service.ui.tree.ITree;
import com.sapportals.portal.prt.service.ui.tree.ITreeNode;
```



```

public class HardCodedTreeRenderer extends
AbstractTreeBuilder {

    protected void initResources(IPortalComponentRequest
request) throws MissingResourceException {
        // TODO Auto-generated method stub
    }

    public ITree getChildren(String nodeID,
IPortalComponentRequest request)
        throws TreeRendererException {
        // TODO Auto-generated method stub
        return null;
    }

    public ITree buildTree(IPortalComponentRequest request)
throws TreeRendererException {
        // TODO Auto-generated method stub
        return null;
    }

    protected void setNodeIcons(ITreeNode
newNode, IBasicNode bObject, IPortalComponentRequest request)
{
        // TODO Auto-generated method stub
    }
}

```

2) Create constructor :

```

public HardCodedTreerenderer(String treeID, String root) {
    super(treeID,root);
}

```

treeID - tree ID

root - it can be a file system path ,XML file name , system alias or other value which can represent the tree root or can be used to retrieve the root and its childrens.

You can specify as more parameters as you want but you must call the super constructor with treeid and root value. Additional parameters can be stored in [mm_dynamicLoadAttributes](#) hashtable for further processing.

3) Add resources initialization:

Add icon declarations at the top of the class, for example :

```

// show closed folder icon
private AbstractTreeRenderer.Icon mm_iconFolderClosed;
// show opened folder icon
private AbstractTreeRenderer.Icon mm_iconFolderOpen;
// show document icon
private AbstractTreeRenderer.Icon mm_iconDocument;

```

Implement **initResources** function which should initialize declared icons with suitable image resource from images directory (you can also specify another directory):

```
protected void initResources(IPortalComponentRequest
request) throws

MissingResourceException {
    mm_iconFolderOpen = new Icon(initResource(request,
IResource.IMAGE, "images/folderopen.gif"));
    mm_iconFolderClosed = new Icon(initResource(request,
IResource.IMAGE, "images/folder.gif"));
    mm_iconDocument= new Icon(initResource(request,
IResource.IMAGE, "images/object.gif"));
}
```

Implement **setNodeIcons** function which defines which icon relate to each the node :

```
protected void setNodeIcons(ITreeNode newNode, IBasicNode
bObject,

IPortalComponentRequest request) {
if (bObject.isFolder()) {

newNode.setFolderOpenImage(mm_iconFolderOpen.getUrl(request)
);
    newNode.setFolderCloseImage(mm_iconFolderClosed.getUrl(
request));
} else {
    // attribute
    newNode.setDocumentImage(mm_iconDocument.getUrl(request
));
}
}
```

- 4) Implementing two most important functions : **buildTree** and **getChildren** which defines the tree structure :

buildTree – this function creates the Root node and adds himself to the tree. You can add the node to the tree by calling **addTreeNode(parentNode, myTreeNode, tree,request)** function that placed in **AbstractTreeBuilder** class.

This is simple example provided from par file attached :

```
public ITree buildTree(IPortalComponentRequest request)
throws

TreeRendererException {
    ITree tree =
AbstractTreeRenderer.getUiService().createTree(getTreeID(),
"");
    tree.setSelectionMode(ITree.SELECTIONMODE_MULTI);
}
```

```

    IBasicNode parentBObject = null;
    // Create the root node
    parentBObject = getNode(null);
    // Add the root to the tree.
    addTreeNode(null, parentBObject, tree, request);

    return tree;
}

```

getChildren - this function creates the child nodes of supplied node (according to its Node ID) and adds the childs to the parent node.

This is simple example provided from par file attached :

```

public ITree getChildren(String nodeID,
    IPortalComponentRequest request) throws
    TreeRendererException {
    ITree tree =
    AbstractTreeRenderer.getUiService().createTree(getTreeID(),
    "");
    tree.setSelectionMode(ITree.SELECTIONMODE_MULTI);

    ITreeNode parentNode = null;
    IBasicNode parentBObject = null;

    IBasicNode[] allChildren = null;

    parentBObject = getNode(nodeID);
    parentNode = addTreeNode(null, parentBObject, tree,
    request);

    allChildren = getChildren(parentBObject.getId());

    for (int i = 0; i < allChildren.length; i++) {
        IBasicNode bObject = allChildren[i];
        addTreeNode(parentNode, bObject, tree, request);
    }
    return tree;
}

```

3.4 Calling the tree renderer class from the portal component.

A class that implements this type of functionality is contained in the example attached.

Steps for calling the tree renderer:

- 1) Init your renderer class at the start of the doContent method , it is better for better performance to initialize the renderer only once, for example:

```
renderer = new HardCodedTreerenderer(TREE_ID, inputVal);
```

- 2) Get the default root of the tree for dynamic loading of the first tree layer .

```
String parentNodeId =  
data.getAttribute(AbstractTreeRenderer.ATTRIBUTE_PARENT_NODE  
_ID);
```

- 3) Call the **refreshTree** function for initializing the first tree layer

```
treeRenderer.refreshTree(parentNodeId, request, response);
```

- 4) Create tree view component by calling the **createInitialTreeHtml** function :

```
HTMLFragment tree = new  
HTMLFragment(treeRenderer.createInitialTreeHtml(req, resp));
```

3.4.1 Sample Code

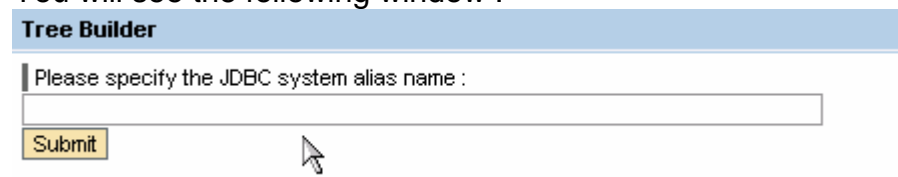
A classes that implements functionality described above is contained in the Appendix. By using the classes provided in example the developer from some of the complexity of getting a connection to JDBC system , perform basic operations on the tree and creating the portal component using the tree render implementation. Attached example contain two tree implementation :

- 1) Building the database structure tree: In order to run this example you need to create JDBC System and assign it a system alias. You can also see how the database connection established in ConnectionUtil class .
- 2) Building the tree from txt file which describes the tree hierarchy.

Prerequisite for running Business Object tree renderer example : JDBC System created:

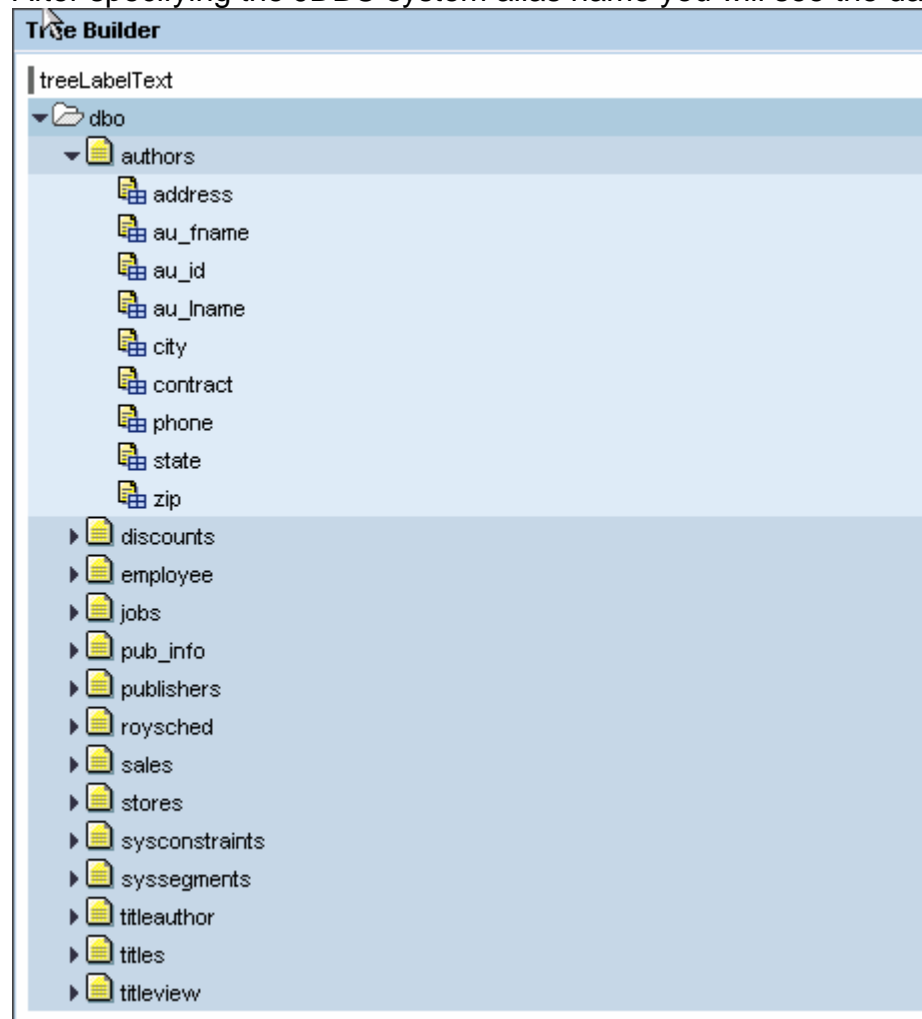
- Upload TreeBuilder par to the portal.
- You can view the tree either by creating an iView from the par or browsing to SAP Enterprise Portal 6.0 Launcher Component and launching **TreeBuilder** component .

You will see the following window :



The screenshot shows a web form titled "Tree Builder". The form has a blue header bar with the text "Tree Builder". Below the header, there is a text input field with the placeholder text "Please specify the JDBC system alias name :". Below the input field, there is a "Submit" button. A mouse cursor is visible over the "Submit" button.

After specifying the JDBC system alias name you will see the database tree:



- To open the TreeBuilder component in Eclipse - Import the project from PAR file and you will see the Tree Render implementation.

4 Risks

- Some versions of Netscape Navigator browsers will not support the dynamic tree.
- Risk of connection timeout if loading the children nodes takes too long.
- Performance may be limited by performance of connectors and performance of Connector Generic component if you component connecting to the database or other applications.

5 Appendix

PAR file contain sample programs (TreeBuilder Portal Component)that utilize DynamicTreeService and builds two types of trees: JDBC tree and tree defined in txt file	See ZIP-File
The text file describing the tree hierarchy. The first number in the line defines the parent node id and the second number in the row defines its child node id.	See ZIP-File

www.sdn.sap.com/irj/sdn/howtoguides