Creating Tree IN Web Dynpro for ABAP - Tutorial.

Applies to:

Web Dynpro for ABAP, NW 7.0. For more information, visit the Web Dynpro ABAP homepage.

Summary

In this tutorial I explain how to create the TREE UI element and changing its structure dynamically at runtime. Therefore you should have some basic knowledge of Web Dynpro ABAP and ABAP OO.

Author: Patrick Johann

Company: SAP

Created on: 17. March 2009

Author Bio

Patrick Johann is in the first year of Vocational Training as an IT Specialist – Application Development at SAP. At the moment he is working in a SAP Net Weaver Product Management User Interaction Team and is doing Web Dynpro for ABAP.

Table of Contents

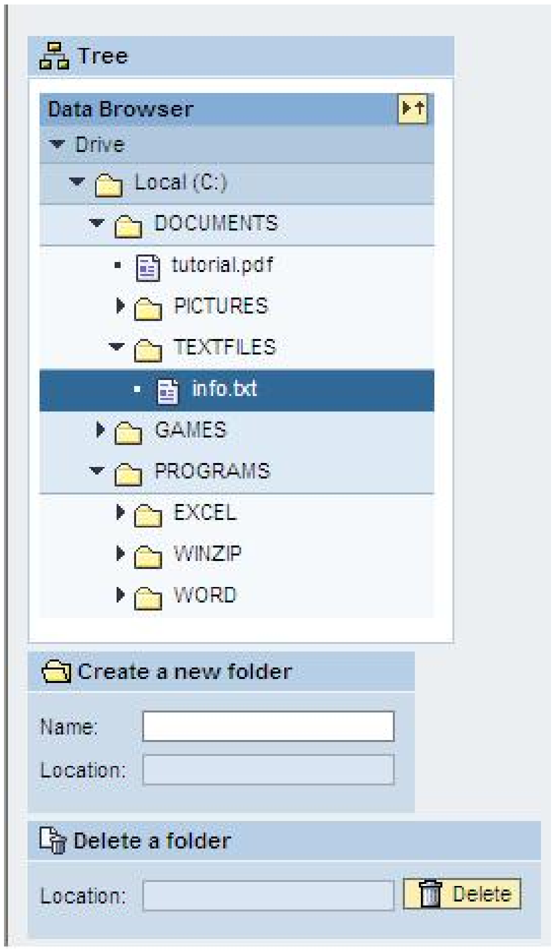
Exercise Objectives: 3
Process at runtime: 3
Procedure - Part A: Tree 4
Procedure – Part B: Create and delete Folders at runtime 6
    onactionload_children: 6
    onactionselect: 7
    tree-fill: 7
    create_node: 8
    wddoinit: 9
    onactioncreate_folder: 9
Related Content 11
Copyright 12

Exercise Objectives:

- Define Trees statically and define a dynamic and efficient binding at runtime

Process at runtime:

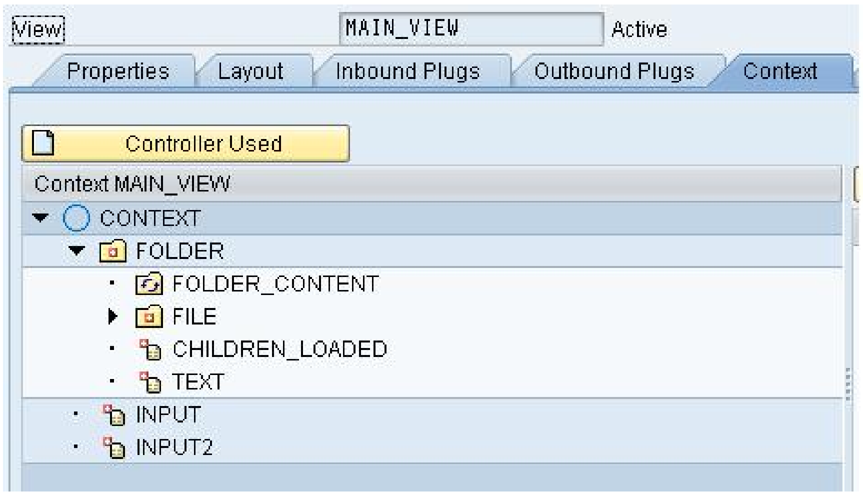
The user navigates through a data browser. First he or she expands the local drive (C:), then the user can navigate between different folders and files. It is possible to create new folders or delete existing ones, too. The items will be displayed alphabetically and in capital letters.



Procedure - Part A: Tree

(For source code look at the ABAP coding below)

1. Start the Object Navigator (transaction code SE80). Create a new Web Dynpro component (suggested name: *'Z##_TREE'*), one window and one view (suggested names: *'MAIN_VIEW'* and *'MAIN_WINDOW'*).
2. The complete coding and context mapping will be done in the view controller instead of the COMPONENTCONTROLLER because we don't need data from the ABAP Dictionary in this tutorial. Create the following nodes and attributes:



Node/Attribute:	Name:	Property:	Other values:
Node	**FOLDER**	**Cardinality 0:n**	**no**
Recursion Node	**FOLDER_CONTENT**	**Repeated node: MAIN_VIEW.FOLDER**	**-**
Node	**FILE**	**Cardinality 0:n**	**no**
Attribute	**TEXT**	**TYPE *string***	**-**
Attribute	**CHILDREN_LOADED**	**TYPE *wdy_boolean***	**-**
Attribute	**TEXT**	**TYPE *string***	**-**
Attribute	**INPUT**	**TYPE *string***	**Help mode: deactivated**
Attribute	**INPUT2**	**TYPE *string***	**Help mode: deactivated**

3. Before we start coding you should design the layout of your Web Dynpro and create the UI element tree. Add three groups to your ROOTUIELEMENTCONTAINER. The first group shall display data browser; therefore insert a tree element. Bind the dataSource to the created context node FOLDER. Moreover you have to insert a TreeNodeType and a TreeItemType to your Tree, because nodes may or may not have child nodes items and we need both types. Bind FOLDER as dataSource at FOLDER.TEXT as text property to the TreeNodeType element. For TreeItemType bind FILE and FILE.TEXT for the same properties. The TreeNodeType has the events onLoadChildren and onAction, create events for those (suggested names: 'LOAD_CHILDREN' and 'SELECT').

4. We will now start coding the ONACTIONLOAD_CHILDREN method. In the coding of this event we reference the context element in the methods signature and check if the child node is already loaded or not. If no child has been loaded a method named CREATE_NODE will be called (we will code it later). Reference to the context element and get the attribute CHILDREN_LOADED. Then check if it is false. If it is, set it to true. After this, get the attribute TEXT and write it to a local defined string variable. Hand it over together with the context element reference as an EXPORTING parameter of the CREATE_NODE method call.

5. The coding for the ONACTIONSELECT method is really short and easy. We read the attribute TEXT and write its value into out INPUT2 attribute. Use the get_attribute( ) and set_attribute( ) method calls to get and set the value.

6. Create two context attributes (suggested names: folder_struc and file_struc). Both are tables of the type TIHTTPNVP. These tables will save the complete structure of our data browser.

7. Our tree won't work until we have filled it with data. Although the user will be able to create folders at runtime, there should be a default file system pre-defined. Create the method TREE_FILL. Define two local variables (one for the files and one for the folders) of type IHTTPNVP. Make some entries in your tree. Your local types have the parameters name (for the name of your file or folder) and value (for the parent folder). After each declaration assign the folders to your context attribute table you created in the previous step and sort the table. You have to assign the files to the other table. I create a windows like data browser with the local drive (C:) as parent key node. ;)

8. Now we start coding the *Create_Node( )* method. This method reads the data from your tables and writes it to the context attribute which fills the tree at runtime. First add two parameters to the method signature:

Element	**IMPORTING/TYPE REF TO**	**If_wd_context_element**
Key	**IMPORTING/TYPE**	**string**

In addition declare a reference to the context node and context element and declare, two local variables of type IHTTPNVP as in step 7. Then read the data from the tables by getting access to the required context nodes (FOLDER_CONTENT and FILE) and create and bind a new element with the instance methods *create_element( )* and *bind_element( )* from your node reference. You have to supply the parameters *new_item* with your local context element reference and *set_initial_elements* with *abap_false* (because we want to add an element) for *bind_element( )*. For reading data use a LOOP AT statement at the table into your IHTTPNVP attribute with the imported key. Inside the LOOP you have to call the method set_attribute( ) to modify the TEXT context attribute. Read the fitting text from your declared table type's field 'name'. Repeat the whole step for the second node.

9. The data browser won't yet have content if you create and start your Web Dynpro application now , because the TREE_FILL method has never been called. Navigate to the hook method WDDOINIT and implement the TREE_FILL method. But we have to do more things in addition. Moreover we have to create our first element and node. This will be our starting navigation node in the data browser. Reference to context node and context element. Then navigate to the FOLDER node with get_child_node. After that create an element and bind the element as in step 8. Furthermore set the lead selection to the created element, because we don't initialize lead selection at context mapping. Set the TEXT attribute to your starting Folder (I've called it "Local (C :)"). Finally implement the *Create_Node* method, hand over the context element reference and your starting folder as key.

Procedure – Part B: Create and delete Folders at runtime

(For source code look at the abap-coding below)

1. Choose the layout tab. The application would run at the moment. You can click through the pre defined data browser, but we want to create a more user friendly design with icons for items and folders. Click on the tree UI element and change the iconSource property of the NodeType and ItemType (suggested icons: Node -> ~Icon/FolderFile , Item -> ~Icon/DocumentFile). You can write a meaningful rootText and title under the tree properties, too. Set the width of the tree to at least 200px for a better view.
2. Insert two input fields in the second group and one in the third. With these input fields you may add or delete folders in your tree. First we work on the second group, which adds new folders. Bind the first input field to the INPUT context attribute and the second one to INPUT2. Set the property of the second field to readOnly. Label the UI elements (suggested names: Label for INPUT and Location for INPUT2).
3. Create a new *onEnter* event for the first input field called CREATE_FOLDER. Navigate to the method behind this event. At first we have to make a couple of declarations for this method. We need a reference to the context node and to the context element, as well as to the two INPUT attributes, because we need the entered name and the location of the folder that has to be created. Then we need a table of type IHTTPNVP again, to append the new folder to the table. Furthermore a structure of type wd_this->element_folder has to be declared, because the INPUT attributes are like a column of this structure.
4. Call the method *wd_context->get_element( )* to get the element of the lead selection. Moreover call the method *get_attribute( )* twice, for INPUT and INPUT 2. Set the value from INPUT to upper case, because the folders shall display in capital letters. Hand over the values from the input fields to the fields name and value of your local IHTTPNVP type. After this you do probably the same as in the WDDOINIT method. We append it to the table and sort it, then we create and bind a element set the lead selection and call the *create_node* method. We build up the whole data browser again. Set the INPUT attributes to an initial value, so the input fields get cleared after creating a folder.
5. To delete folders we create the layout of the third group first. The group should have one input field, one label and one button. Set the input field to *readOnly*, so that the user can't make an input in this field. The field will be filled via selected item. Then he or she has to click on the button and the folder will be deleted. Create the *onAction* event for the button called DELETE_FOLDER and start coding the method. This method runs through the same procedure as the method for creating a folder. The difference is that you only need access to one context attribute, because we only have one input field (INPUT2, which displays the parent key) and you have to delete an entry instead appended to the table. For this use the DELETE statement and delete an entry from the *folder_struc* table where the name property is the same as the value from your INPUT attribute. Moreover you can generate a message via Code Wizard if the selected item was the root folder (it will always be created new, so it can't be deleted).

onactionload_children:

```
METHOD onactionload_children .

  DATA: key                 TYPE    string,
        children_loaded     TYPE    wdy_boolean.


  context_element->get_attribute(
    EXPORTING name = 'CHILDREN_LOADED'
    IMPORTING value = children_loaded ).

  CHECK children_loaded = abap_false.

  context_element->set_attribute(
  EXPORTING name = 'CHILDREN_LOADED' value = abap_true ).

  context_element->get_attribute(
  EXPORTING name = 'TEXT'
    IMPORTING value = key ).

  create_node( EXPORTING element = context_element key = key ).

ENDMETHOD.
```

onactionselect:

```
METHOD onactionselect .

  DATA   lo_text   TYPE   string.

  context_element->get_attribute(
    EXPORTING name = 'TEXT'
    IMPORTING value = lo_text ).

  wd_context->set_attribute(
    EXPORTING name = 'INPUT2' value = lo_text ).

ENDMETHOD.
```

tree-fill:

```
METHOD tree_fill .

  DATA:         lo_folder     TYPE    ihttpnvp,
               lo_file       TYPE    ihttpnvp.

  lo_folder-name = 'DOCUMENTS'.
  lo_folder-value = 'C'.
  APPEND lo_folder TO wd_this->folder_struc.

  lo_folder-name = 'PROGRAMS'.
  lo_folder-value = 'C'.
  APPEND lo_folder TO wd_this->folder_struc.

  lo_folder-name = 'GAMES'.
  lo_folder-value = 'C'.
  APPEND lo_folder TO wd_this->folder_struc.

  lo_folder-name = 'TEXTFILES'.
  lo_folder-value = 'DOCUMENTS'.
  APPEND lo_folder TO wd_this->folder_struc.

  lo_folder-name = 'PICTURES'.
  lo_folder-value = 'DOCUMENTS'.
  APPEND lo_folder TO wd_this->folder_struc.

  lo_folder-name = 'EXCEL'.
  lo_folder-value = 'PROGRAMS'.
  APPEND lo_folder TO wd_this->folder_struc.

  lo_folder-name = 'WINZIP'.
  lo_folder-value = 'PROGRAMS'.
  APPEND lo_folder TO wd_this->folder_struc.

  lo_folder-name = 'WORD'.
  lo_folder-value = 'PROGRAMS'.
  APPEND lo_folder TO wd_this->folder_struc.

  lo_folder-name = 'PINGPONG'.
  lo_folder-value = 'GAMES'.
  APPEND lo_folder TO wd_this->folder_struc.
  SORT wd_this->folder_struc.
*********************************FILES****************************
******************************************************************
  lo_file-name = 'setup.exe'.
  lo_file-value = 'GAMES'.
  APPEND lo_file TO wd_this->file_struc.

  lo_file-name = 'uninstall.exe'.
  lo_file-value = 'WINZIP'.
  APPEND lo_file TO wd_this->file_struc.

  lo_file-name = 'pic01.jpg'.
  lo_file-value = 'PICTURES'.
  APPEND lo_file TO wd_this->file_struc.

  lo_file-name = 'pic02.jpg'.
  lo_file-value = 'PICTURES'.
  APPEND lo_file TO wd_this->file_struc.

  lo_file-name = 'tutorial.pdf'.
  lo_file-value = 'DOCUMENTS'.
  APPEND lo_file TO wd_this->file_struc.

  lo_file-name = 'info.txt'.
  lo_file-value = 'TEXTFILES'.
  APPEND lo_file TO wd_this->file_struc.

ENDMETHOD.
```

create_node:

```
METHOD create_node .

  DATA:       lo_node      TYPE REF TO   if_wd_context_node,
             lo_element   TYPE REF TO   if_wd_context_element,
             folder       TYPE          ihttpnvp,
             file         TYPE          ihttpnvp.

  lo_node = element->get_child_node( name = 'FOLDER_CONTENT').

  LOOP AT wd_this->folder_struc INTO folder WHERE value = key.

    lo_element = lo_node->create_element( ).
    lo_element->set_attribute(
    EXPORTING name = 'TEXT' value = folder-name ).
    lo_node->bind_element( new_item = lo_element set_initial_elements = abap_false ).

  ENDLOOP.

  lo_node = element->get_child_node( name = 'FILE').

  LOOP AT wd_this->file_struc INTO file WHERE value = key.

    lo_element = lo_node->create_element( ).
    lo_element->set_attribute(
    EXPORTING name = 'TEXT' value = file-name ).

    lo_node->bind_element( new_item = lo_element set_initial_elements = abap_false ).

  ENDLOOP.

ENDMETHOD.
```

wddoinit:

```
METHOD wddoinit .

  DATA:     lo_node        TYPE REF TO if_wd_context_node,
           lo_element     TYPE REF TO if_wd_context_element.

  tree_fill( ).

  lo_node = wd_context->get_child_node( name = 'FOLDER' ).
  lo_element = lo_node->create_element( ).
  lo_node->bind_element( lo_element ).
  lo_node->set_lead_selection( lo_element ).
  lo_element->set_attribute( name = 'TEXT' value = 'Local (C:)' ).

  create_node( EXPORTING element = lo_element  key = 'C' ).

ENDMETHOD.
```

onactioncreate_folder:

```
METHOD onactioncreate_folder .

  DATA:  lo_element       TYPE REF TO if_wd_context_element,
         lo_node          TYPE REF TO if_wd_context_node,
         ls_input         TYPE        wd_this->element_context,
         lv_input         LIKE        ls_input-input,
         lv_input2        LIKE        ls_input-input2,
         folder           TYPE        ihttpnvp.

  lo_element = wd_context->get_element( ).

  lo_element->get_attribute(
    EXPORTING name = 'INPUT'
    IMPORTING value = lv_input ).

  lo_element->get_attribute(
    EXPORTING name = 'INPUT2'
    IMPORTING value = lv_input2 ).
  TRANSLATE lv_input TO UPPER CASE.

  folder-name = lv_input.
  folder-value  = lv_input2.
  APPEND folder TO wd_this->folder_struc.

  SORT wd_this->folder_struc.

  lo_element->set_attribute( EXPORTING name = 'INPUT' value = ' ' ).
  lo_element->set_attribute( EXPORTING name = 'INPUT2' value = ' ' ).

  lo_node = wd_context->get_child_node( name = 'FOLDER' ).
  lo_element = lo_node->create_element( ).
  lo_node->bind_element( lo_element ).
  lo_node->set_lead_selection( lo_element ).
  lo_element->set_attribute( EXPORTING name = 'TEXT' value = 'Local (C:)' ).

  create_node( EXPORTING element = lo_element key = 'C' ).

ENDMETHOD
```

Related Content

For more information, visit the Web Dynpro ABAP homepage.

For more information, visit the User Interface Technology homepage.

Copyright