Exploring Table Popin in Web Dynpro for Java

Applies to:

Web Dynpro for Java applications for SAP Netweaver 04 SP Stack 17, SAP Netweaver 04s SP Stack 15. For more information, visit the User Interface Technology Homepage.

Summary

This article demonstrates the use of TablePopin UI element within the Table UI element in Web Dynpro for Java. It illustrates the implementation of both the variants of TablePopin UI element: Row Popin and Column Popin.

Author: Divyata Dal

Company: Larsen and Toubro Infotech Limited

Created on: 17 March 2010

Author Bio

Divyata Dal is working at Larsen and Toubro Infotech Limited as a SAP ABAP and SAP Netweaver Developer for Web Dynpro in Java. She has knowledge in diverse aspects of ABAP, Web Dynpro for Java, SAP Enterprise Portal, Java/J2EE and Adobe Flex.

Table of Contents

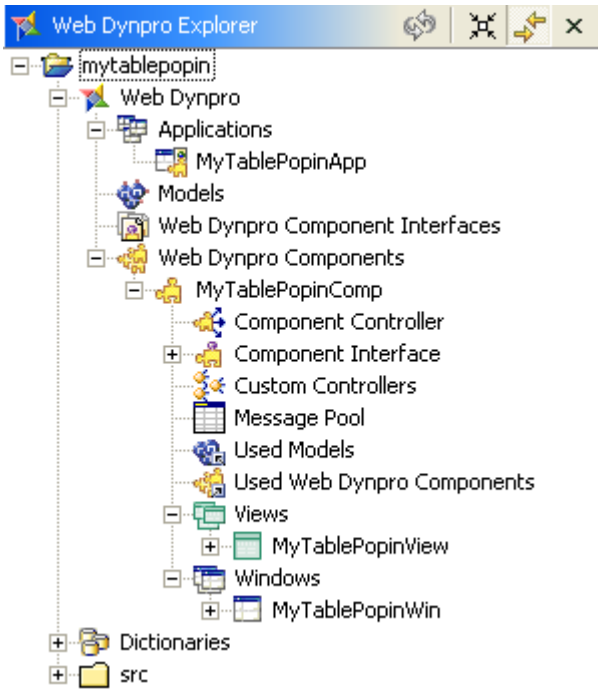
Introduction

Table Popins are an add-on to the Table UI element. They can be used to dynamically display further information about a particular row or a table cell. They are insertions between the rows of a Table, referring to either the entire row of the table or an individual table column. A popin can be linked to the table row for a *Row Popin* and to the column for a *Column Popin* or *Cell Popin*.

We can have only one row popin per table, but multiple column popins. The information associated with a particular table cell can be displayed using the Column Popin, whereas to display the data relevant to a particular row of the table the Row Popin can be used. How these two variants of the Table Popin UI element can be implemented will be explained at length in this article.

Creating a Web Dynpro application

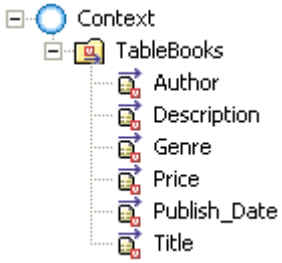
Create a new Web Dynpro DC named **mytablepopin** with the component named MyTablePopinComp and the application named MyTablePopinApp. Let the window be named MyTablePopinWin and its view MyTablePopinView.



Creating a Context for the Table Data and Mapping it to the View Context

In order to manipulate the data in the table, we will create a value node in the *View*.

1. Create a value node called ‘TableBooks’. The cardinality of this node would be default value of [0…n]. Change the selection cardinality to [0…n].
2. Now add the attributes ‘Author’, ‘Title’, ‘Genre’, ‘Price’, ‘Publish_Date’, ‘Description’ to this node and set their data type as String.
3. The ‘TableBooks’ node of the view should now look like this:

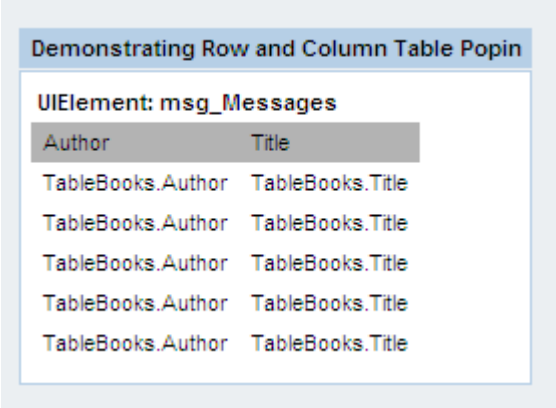


4. Add another attribute ‘No_of_Rows’ of type *integer* to the context, to indicate the number of entries in the table.

Designing the View

In order to display information on the screen to demonstrate use of the Table Popin, we shall now proceed with designing the view.

1. Switch to the Layout tab of the view and delete the default TextView element.
2. Right click on the Root Element and add a Group it, grp_Main and set the header to “Demonstrating Row and Column Table Popin”.
3. Now right-click on the group and add a Table UI element to the group, tbl_TableBooks. This table must be mapped to a data source. For this, right-click on the table and select *Apply Template*.
4. Select the attributes ‘Author’ and ‘Title’ and click *Next*.
5. Change the value of ‘Editor’ column for *Title* to LinkToAction.
6. Confirm with *Finish*.



5. Now, switch to the Properties of the table and change the *selectionMode* property of the table from *auto* to *none*.
6. Map the table property *visibleRowCount* to the context attribute *No_of_Rows*.

Populating Data in the Table

In order to demonstrate the data in the table, we first have to populate the table with some dummy data.

1. Switch to the *Methods* tab of the view and add a new method mPopulateTable to it.
2. We shall use an XML file to populate the table (say, Books.xml).
3. Navigate to the *Implementation* section and insert the following code in the method just created:

```
IPrivateMyTablePopinView.ITableBooksNode l_tblNode =
                                                    wdContext.nodeTableBooks();
IPrivateMyTablePopinView.ITableBooksElement l_tblEle = null;
   String l_strXMLPath = null;

   try
   {
l_strXMLPath = WDURLGenerator.getResourcePath(
wdComponentAPI.getDeployableObjectPart(),
                                    "Books.xml");
          FileReader l_xmlTemplate = new FileReader(l_strXMLPath);
          InputSource l_inputSource = new InputSource(l_xmlTemplate);
          DocumentBuilderFactory l_dbf = DocumentBuilderFactory.newInstance();
          DocumentBuilder l_db = l_dbf.newDocumentBuilder();
          Document l_xmlDoc = null;
          l_xmlDoc = l_db.parse(l_inputSource);
          NodeList XML_BookNode = l_xmlDoc.getElementsByTagName("book");
          for(int i = 0; i < XML_BookNode.getLength(); i++)
          {
                 Node XML_Node = XML_BookNode.item(i);
                 l_tblEle = l_tblNode.createTableBooksElement();
                 l_tblEle.setAuthor(XML_Node.getChildNodes().item(1)
.getChildNodes().item(0).toString());
                 l_tblEle.setTitle(XML_Node.getChildNodes().item(3)
.getChildNodes().item(0).toString());
                 l_tblEle.setGenre(XML_Node.getChildNodes().item(5)
.getChildNodes().item(0).toString());
                 l_tblEle.setPrice(XML_Node.getChildNodes().item(7)
.getChildNodes().item(0).toString());
                 l_tblEle.setPublish_Date(XML_Node.getChildNodes().item(9)
.getChildNodes().item(0).toString());
                 l_tblEle.setDescription(XML_Node.getChildNodes().item(11)
.getChildNodes().item(0).toString());
                 l_tblNode.addElement(l_tblEle);
          }

          wdContext.currentContextElement().setNo_of_Rows(l_tblNode.size());
   }
   catch(FileNotFoundException e)
   {
          wdComponentAPI.getMessageManager()
.reportException("FileNotFoundException "+e.getMessage(),
                       true);
   }
   catch(FactoryConfigurationError e)
   {
          wdComponentAPI.getMessageManager()
.reportException("FactoryConfigurationError "+e.getMessage(),
```

```
true);
   }
   catch(SAXException e)
   {
          wdComponentAPI.getMessageManager()
.reportException("SAXException "+e.getMessage(),
true);
   }
   catch(ParserConfigurationException e)
   {


wdComponentAPI.getMessageManager()
.reportException ("ParserConfigurationException "+
e.getMessage(), true);
   }
   catch(WDAliasResolvingException e)
   {
          wdComponentAPI.getMessageManager()
.reportException ("WDAliasResolvingException "
+e.getMessage(), true);
}
   catch(IOException e)
   {
          wdComponentAPI.getMessageManager()
.reportException("IOException "+e.getMessage(), true);
          }
```

4. This method must be called in the *wdDoInit()* method of the view.
5. Now, Save, 'Deploy New Archive and Run'

Demonstrating Row and Column Table Popin

Author	Title
Gambardella, Matthew	XML Developer's Guide
Ralls, Kim	Midnight Rain
Corets, Eva	Maeve Ascendant
Corets, Eva	Oberon's Legacy
Corets, Eva	The Sundered Grail
Randall, Cynthia	Lover Birds
Thurman, Paula	Splish Splash
Knorr, Stefan	Creepy Crawlies
Kress, Peter	Paradox Lost
O'Brien, Tim	Microsoft .NET: The Programming Bible
O'Brien, Tim	MSXML3: A Comprehensive Guide
Galos, Mike	Visual Studio 7: A Comprehensive Guide

Implementing Row Popin for the Table

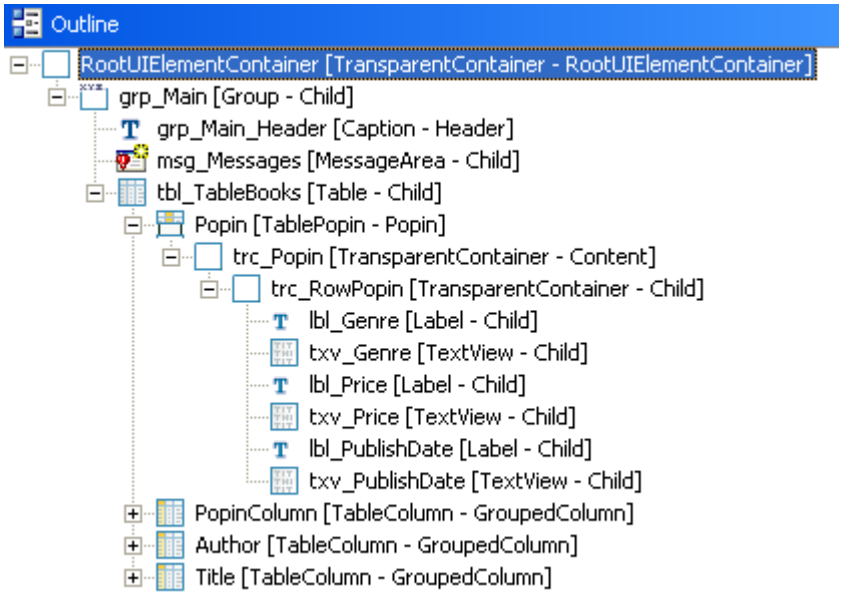
A Row Popin is linked to a table row. The content of a Popin can be any UI element as per the user requirement. In our example, we shall display the 'Genre', 'Price' and 'Publish_Date' corresponding to each book of the table row.

Adding a Row Popin to the Table UI Element

The Table Popin UI element has to be embedded within the table created. This can be achieved by the following steps:

1. Right click on the table 'TableBooks' in the Layout tab of view and select *Insert Popin*. Let its ID be 'Popin'.
2. Right click on the Popin and add a transparent container 'trc_RowPopin'
3. To the container 'trc_RowPopin' add 3 labels and 3 text views each for *Genre*, *Price* and *PublishDate* fields respectively and map them to the corresponding context attributes in the node *TableBooks*

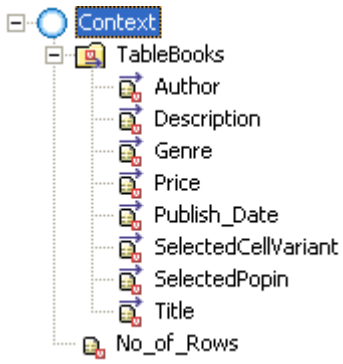
The Layout of the UI elements appears as shown below:



Opening a Popin to Display the Row Details

In order to implement a Row Popin, we will follow the steps as given below:

1. Add two more attributes to the node ‘TableBooks’, namely ‘SelectedPopin’ and ‘SelectedCellVariant’, both of type *String*. The ‘Table.SelectedPopin’ controls whether and, if yes, which popin is displayed. The property must be linked to the DataSource of the table, say TableBooks. The ‘SelectedPopin’ property of the table UI element or column contains the ID of the element whose popin is to be displayed. Either the ID of the table for the row popin or the ID of the column in case of cell popin. If the value of the attribute ‘SelectedPopin’ is set to initial, then no popin is displayed for the row. The context will now look as shown below:



2. Switch to the Layout tab of the view and right click on the Table UI element. Add a new Column ‘PopinColumn’.
3. Since it is mandatory to add a TableCellEditor, right click on the ‘PopinColumn’ and add a TextView, say. Set some text and set the Visibility to *none*.
4. Right click on the ‘PopinColumn’ and select *Insert CellVariant -> TablePopinToggleCell.* This cell variant is used for opening the row popin for each row and is preferably placed at the extreme ends of the table, that is, either as the first column or as the last column.
5. Now for the Event onToggle of the TablePopinToggleCell, create an Action ‘aShowRowPopin’. Add a parameter ‘p_popin’ (note that this must have the same name as the mapping parameter defined in the wdDoModifyView() method) of type ‘IPrivateMyTablePopinView.ITableBooksElement’. (‘p_popin’ is used as a mapping parameter and will be declared in the wdDoModifyView() method in the view)
6. Switch to the *Implementation* tab and add the following piece of code to display the data corresponding to the row for which the popin is selected. Note that, here the value set for the context attribute ‘SelectedPopin’ is the ID of the row popin:

```
IPrivateMyTablePopinView.ITableBooksNode l_tableNode =
wdContext.nodeTableBooks();
IPrivateMyTablePopinView.ITableBooksElement l_tableEle = null;
int l_size = wdContext.nodeTableBooks().size();

for(int i=0 ; i<l_size ; i++)
{
l_tableEle = l_tableNode.getTableBooksElementAt(i);
      if(p_popin.index() == i)
            l_tableEle.setSelectedPopin("Popin");
      else
            l_tableEle.setSelectedPopin("");
}
```

7. Now, the *variantKey* property of the TablePopinToggleCell and the table column PopinColumn's *selectedCellVariant* property must be set to the same value. Set the value of the *variantKey* property to 'CellVariant'. Map the PopinColumn's *selectedCellVariant* property to the table attribute 'TableBooks.SelectedCellVariant'. This attribute will also be set to the value 'CellVariant' in the wdDoInit() method of the view.
8. Now, in order to identify the Popin to be used for this particular table as a Row Popin, we have to map the s*electedPopin* property of the table 'TableBooks' to the 'TableBooks.SelectedPopin' attribute.
9. Now add the following piece of code in the wdDoModifyView() method of the view:

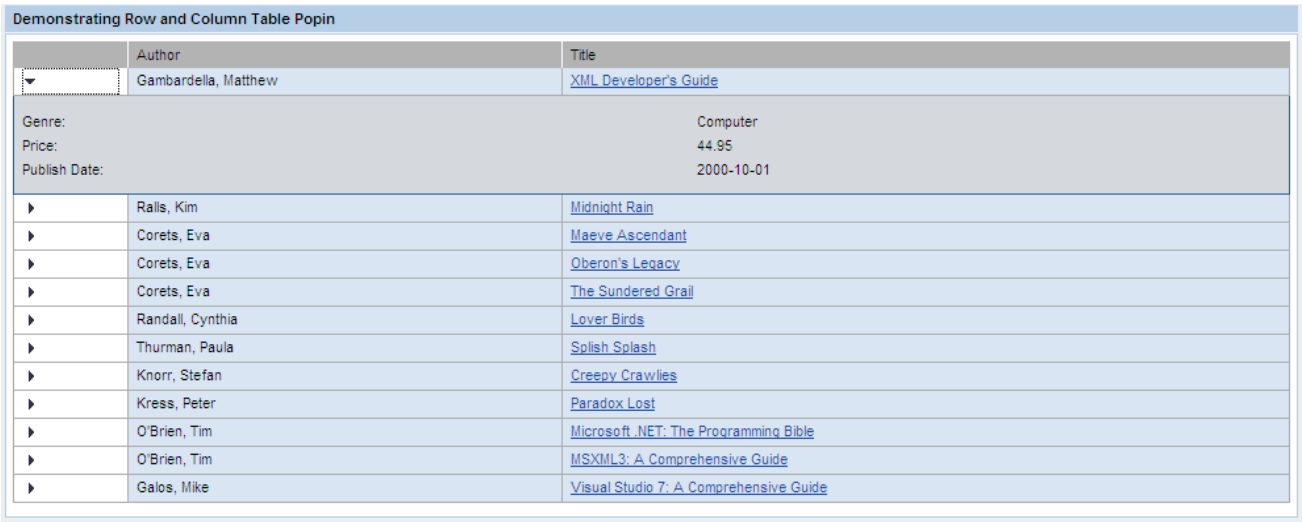
```
if(firstTime)
{
IWDTablePopinToggleCell l_popinCell =
                    (IWDTablePopinToggleCell)
view.getElement("PopinColumn_editor");
             l_popinCell.mappingOfOnToggle()
.addSourceMapping("nodeElement", "p_popin");
}
```

10. Now, initialize the context attributes 'SelectedPopin' and 'SelectedCellVariant' in the wdDoInit() method of the view. The code in the wdDoInit() method now looks as under:

```
mPopulateTable();
IPrivateMyTablePopinView.ITableBooksNode l_tblNode =
                                                wdContext.nodeTableBooks();
IPrivateMyTablePopinView.ITableBooksElement l_tblEle = null;

int l_size = wdContext.nodeTableBooks().size();
for(int i=0 ; i<l_size ; i++)
{
l_tblEle = l_tblNode.getTableBooksElementAt(i);
l_tblEle.setSelectedCellVariant("CellVariant");
l_tblEle.setSelectedPopin("");
}
```

11. Build the project, 'Deploy New Archive and Run'



12. We observe that when we toggle a particular row, the details pertaining to that book are displayed. If we select the toggle button for another row, the previous details are hidden and the popin for the newly selected row are displayed.

Closing the Row Popin

We observe that when we click on a particular toggle cell, the details pertaining to that row are displayed. Now, if we click the toggle cell corresponding to another row, the previous popin is closed, and a new popin is displayed with the details corresponding to the new row.

However, if we wish to close a popin, without opening another one, then we will have to explicitly do so. In general, there are two ways to close a popin

1. Click on the toggle cell of the open popin
   This is relatively simple and we have to simply modify the condition in the action event 'aShowRowPopin'.
   So, `if(p_popin.index() == i)`
   Becomes `if(p_popin.index() == i && l_tableEle.getSelectedPopin() != null)`
   This indicates that the value of the popin is 'null' if it is already open and is selected again
2. Click on the [x] button to the top-right corner of the popin
   This appears only if we have an action for the *onClose* event of the table popin. For this, we will create an action 'aCloseRowPopin' in the view. Add a parameter 'p_row' (note that this must have the same name as the mapping parameter defined in the wdDoModifyView() method) of type 'IPrivateMyTablePopinView.ITableBooksElement'. Map this action to the 'onClose' event of the Popin. The following piece of code must be put in the event:

   Inside the 'firstTime' condition in wdDoModifyView() method:

```
IWDTablePopin l_tblPopin = (IWDTablePopin) view.getElement("Popin");
l_tblPopin.mappingOfOnClose().addSourceMapping("nodeElement", "p_row");
```

   Inside aCloseRowPopin:

```
IPrivateMyTablePopinView.ITableBooksNode l_tableNode =
                                          wdContext.nodeTableBooks();
IPrivateMyTablePopinView.ITableBooksElement l_tableEle = null;

int l_size = wdContext.nodeTableBooks().size();
for(int i=0 ; i<l_size ; i++){
      l_tableEle = l_tableNode.getTableBooksElementAt(i);
      if(p_row.index() == i)
            l_tableEle.setSelectedPopin("");
}
```

The output appears as shown below. Notice the [x] at the top-right corner of the popin content area.

Implementing Column Popin for the Table

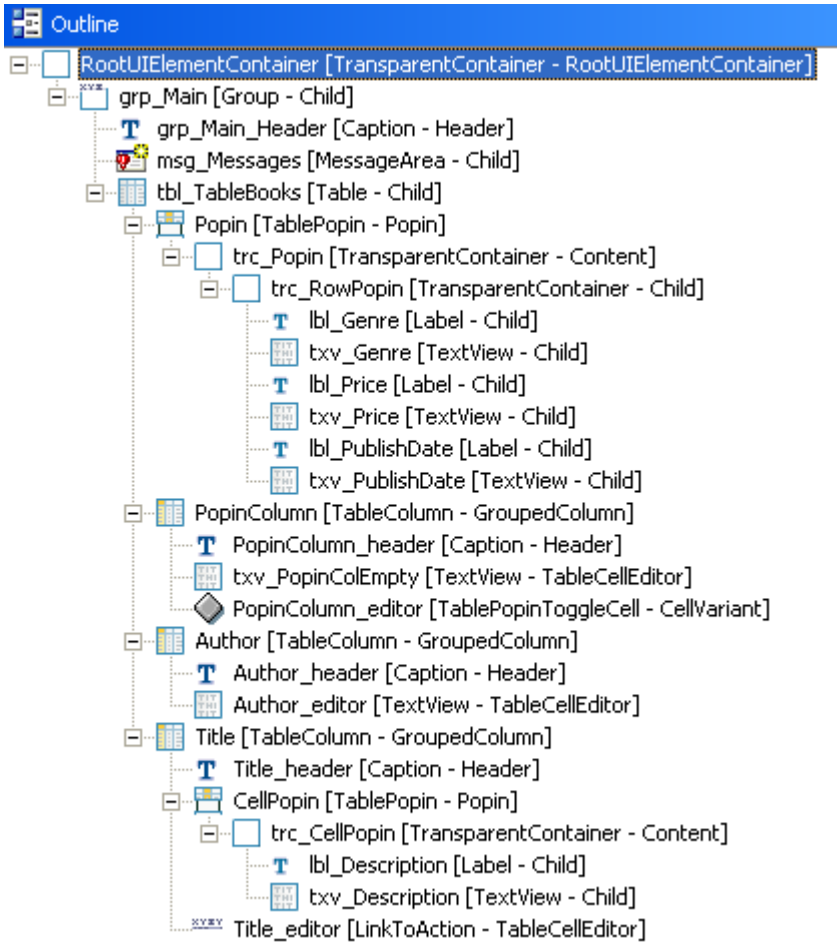
A Column or Cell Popin is added to a column of a table and displays data corresponding to a particular cell of the table. With a Cell Popin, the associated cell is assigned the background color of the popin so that we can see that they belong together. In our example, we shall create a column popin for the 'Title' column of the table 'TableBooks' and display the *Description* of the book on the click of the popin.

Adding a Popin to the Table UI Element as Column Popin

In case of a Column Popin, we would add a Popin to the Column 'Title' using the following steps:

1. Right click on the column 'Title' and select 'Insert Popin'. Set the ID of the popin as *CellPopin.*
2. Right click on the popin -> *Insert Content*. Add a TransparentContainer 'trc_CellPopin'.
3. Now, add a Label and a TextView to it, with the Label text as 'Description' and the TextView mapped to the TableBooks.Description context attribute.

The Layout now appears as shown below:



Opening the Column Popin to Display the Cell Details

The column editor for the 'Title' column is a LinkToAction UI element, which is used to display the contents of the cell popin. Unlike the row popin, there is no complexity involved in creating a column popin. The following simple steps need to be followed:

1. Add the parameter mapping for the "row" parameter for the action for the LinkToAction UI element in the wdDoModifyView() method of the view

```
IWDLinkToAction l_ltaPopin = (IWDLinkToAction)
                             view.getElement("Title_editor");
l_ltaPopin.mappingOfOnAction()
.addSourceMapping("nodeElement", "p_cellpopin");
```

2. Switch to the properties of the LinkToAction UI element and create an action event ‘mShowCellPopin’ for the *onAction* event. Create a parameter ‘p_cellpopin’ (note that this must have the same name as the mapping parameter defined in the wdDoModifyView() method) of type ‘IPrivateMyTablePopinView.ITableBooksElement’. Add the following piece of code in the implementation of this event:

```
IPrivateMyTablePopinView.ITableBooksNode l_tableNode =
wdContext.nodeTableBooks();
      IPrivateMyTablePopinView.ITableBooksElement l_tableEle = null;

int l_size = wdContext.nodeTableBooks().size();


      for(int i=0 ; i<l_size ; i++)
      {
            l_tableEle = l_tableNode.getTableBooksElementAt(i);
            if(p_cellpopin.index() == i)
                  l_tableEle.setSelectedPopin("CellPopin");
            else
                  l_tableEle.setSelectedPopin("");
      }
```

3. Build the project, ‘Deploy New Archive and Run’.

Closing the Column Popin

Unlike the Row Popin, there is only one method to close the Cell Popin, identical to the second method followed for the row popin.

1. Add the following parameter mapping corresponding to the cell popin in the wdDoModifyView() method of the view:

```
IWDTablePopin l_cellPopin = (IWDTablePopin) view.getElement("CellPopin");
      l_cellPopin.mappingOfOnClose().addSourceMapping("nodeElement",
"p_cell");
```

2. Switch to the *Layout* tab and define a method 'aCloseCellPopin' for the 'onClose' event of the popin 'CellPopin'. Add a parameter 'p_cell' note that this must have the same name as the mapping parameter defined for the table popin 'CellPopin' in the wdDoModifyView() method) of type 'IPrivateMyTablePopinView.ITableBooksElement'. In the implementation of the method, add the following piece of code:

```
IPrivateMyTablePopinView.ITableBooksNode l_tableNode =
wdContext.nodeTableBooks();
      IPrivateMyTablePopinView.ITableBooksElement l_tableEle = null;
      int l_size = wdContext.nodeTableBooks().size();

      for(int i=0 ; i<l_size ; i++)
{
            l_tableEle = l_tableNode.getTableBooksElementAt(i);
            if(p_row.index() == i)
                  l_tableEle.setSelectedPopin("");
      }
```

3. Build Project, 'Deploy New Archive and Run'. We notice the [x] at the top-right corner of the popin.

Related Content

Demystifying XML Parsing for Web Dynpro Applications

Sample XML File

Table Popin Documentation

Disclaimer and Liability Notice

This document may discuss sample coding or other information that does not include SAP official interfaces and therefore is not supported by SAP. Changes made based on this information are not supported and can be overwritten during an upgrade.

SAP will not be held liable for any damages caused by using or misusing the information, code or methods suggested in this document, and anyone using these methods does so at his/her own risk.

SAP offers no guarantees and assumes no responsibility or liability of any type with respect to the content of this technical article or code sample, including any liability resulting from incompatibility between the content within this document and the materials and services offered by SAP. You agree that you will not hold, or seek to hold, SAP responsible or liable with respect to the content of this document.