Supply and Dispose Function in Web Dynpro for Java

Applies to:

Web Dynpro for Java (SAP Netweaver CE 7.11). For more information, visit the Web Dynpro Java homepage.

Summary

This article explains how to implement supply and dispose functions in Web Dynpro for Java.

Author: Vani Shanmugam

Company: HCL Technologies

Created on: 05 March 2009

Author Bio

Vani Shanmugam is an Application Developer working in HCL Technologies Limited on SAP NetWeaver Technology

Table of Contents

Introduction

This tutorial focuses on supply and dispose functions and illustrates how to implement these functions using a working example. First we will see the brief explanation of these functions.

Supply function

The supply function is used to fill values for a context node. Instead of creating a collection with values and binding it to the node, the programmer specifies the collection by describing the function to be called by the Web Dynpro runtime environment. Any context node may have a supply function defined for it. A dependent Singleton node must have a supply function defined. Supply functions are defined declaratively.

The Web Dynpro runtime environment calls the supply function of a node when its data is needed and its collection is still invalid. This can be the case if:

- The collection or the node was just created.
- The lead selection of the parent node has changed and an attempt to read the child node.
- The node is invalidated using an API call IWDNode.invalidate().

The supply function of a specific node is defined in the corresponding controller and it has the following signature:

void supply(ChildNode node, ParentElement parentElement);

Dispose function

Dispose function is used to save the contents of a singleton node immediately before that collection is destroyed and the supply function is called. This function is declared programmatically and it can be implemented as an inner class of the component controller. A dispose function belonging to a child node is called automatically when the lead selection in the parent node changes.

A dispose function is written as an inner class of the component controller and this class must implement the IWDNodeCollectionDisposer interface. The wdDoInit() should have initiation of the dispose class and the instance is assigned to the metadata of the appropriate context node.

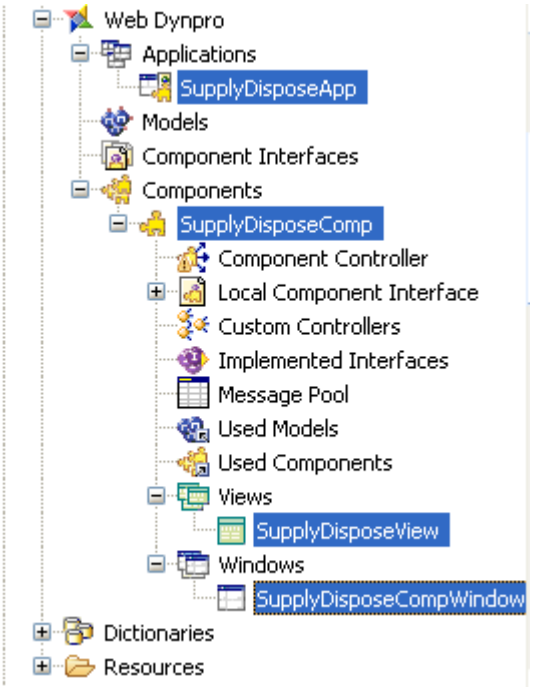
```
public void wdDoInit()
{
   //@@begin wdDoInit()
  // Define the disposer function for the order node
  DisposeOrders orderDisposer = new DisposeOrders();
  wdContext.nodeOrder().getNodeInfo().setCollectionDisposer(orderDisp
  oser);
   //@@end
}


private class DisposeOrders implements IWDNodeCollectionDisposer {
      public void disposeElements(IWDNode node) {
       }

  }
```

The sample application used here is the customer and their orders. Whenever the user selects any customer, the order placed by that customer will be populated in the order table. The user can update any

value in the order table and if he selects some other customer, the updated data will be copied and displayed in the saved orders table. The implementation of suppose and dispose functions are described in the following sections.

Creating a Web Dynpro component

Create a new webdynpro component. The application is named as SupplyDisposeApp, and the component is SupplyDisposeComp.



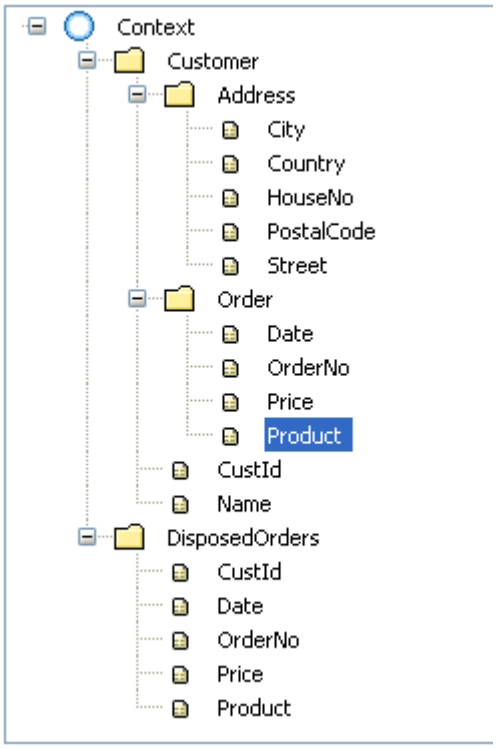
Creating context in the component controller

To provide the data in the table, we need to store the data in the context of the component controller.

- Create an independent node named Customer.
- Create a node named Address to the Customer node (This is optional).
- Create a singleton node named Order to the Customer node by setting the Singleton property to true. Assign the new supply function supplyOrder to the Supply Function property.
- Then add attributes to the Customer and Order nodes.
- Create one more node named DisposedOrders to the context to preserve the Order data. DisposedOrders will have the same set of attributes as the child node Order. In addition to that, we will add one more attribute CustId to the DisposedOrders node.

Node name	Attribute Name	Type
Customer	CustId	integer
Customer	Name	String
Address	City	String
Address	Country	String
Address	HouseNo	String
Address	PostalCode	String
Address	Street	String
Order	Date	String
Order	OrderNo	integer
Order	Price	String
Order	Product	String

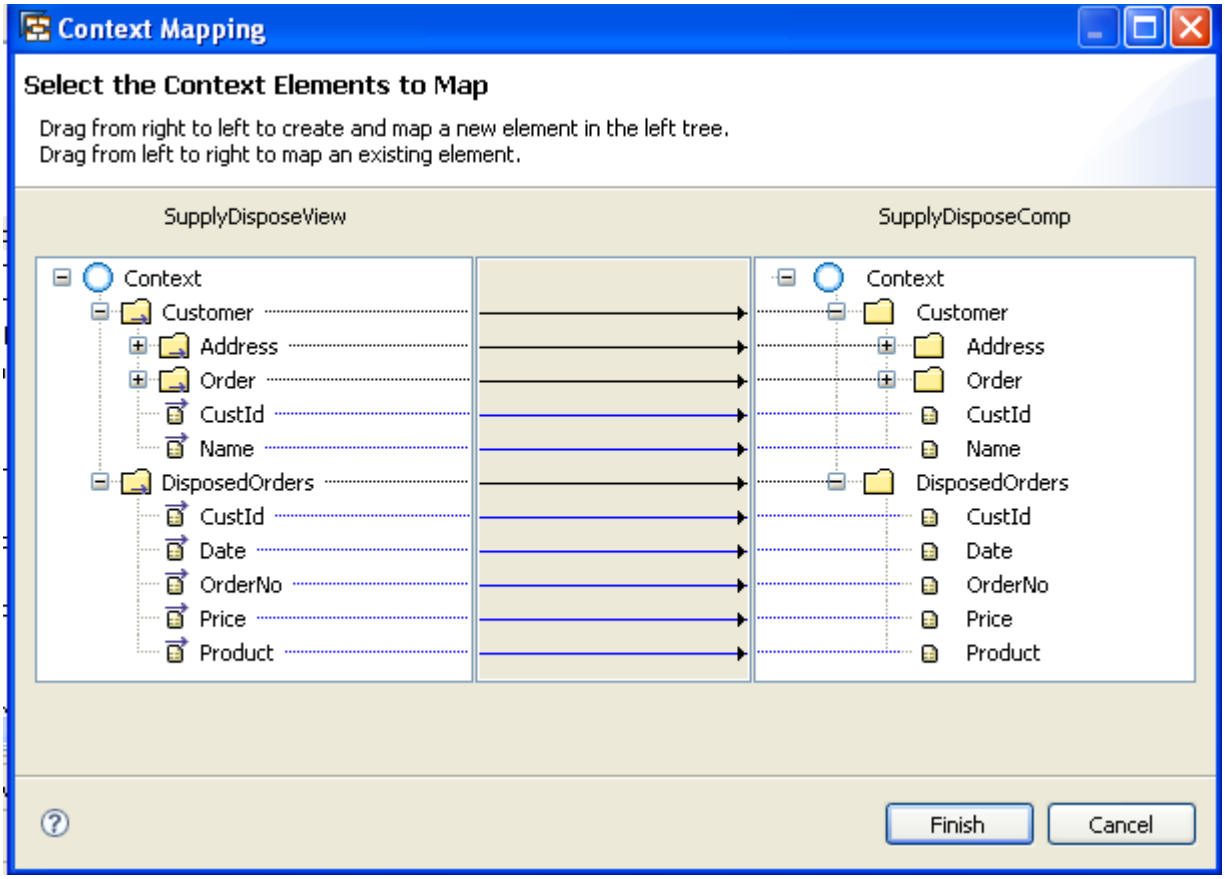
The context of the component controller looks like this


Context
Customer
Address
City
Country
HouseNo
PostalCode
Street
Order
Date
OrderNo
Price
Product
CustId
Name
DisposedOrders
CustId
Date
OrderNo
Price
Product

Creating view context and mapping to the component controller context

To make the data available in the view, we need to map the view context to the component controller context. This can be achieved through context mapping.

- Double click the component SupplyDisposeComp and create a data link between the SupplyDisposeView icon and the Component Controller icon. This causes the component controller to be added to the view controller's list of Required Controllers.
- Drag and Drop the Customer node from the component controller context on the right to the left context node root node. Select all the attributes and choose finish.

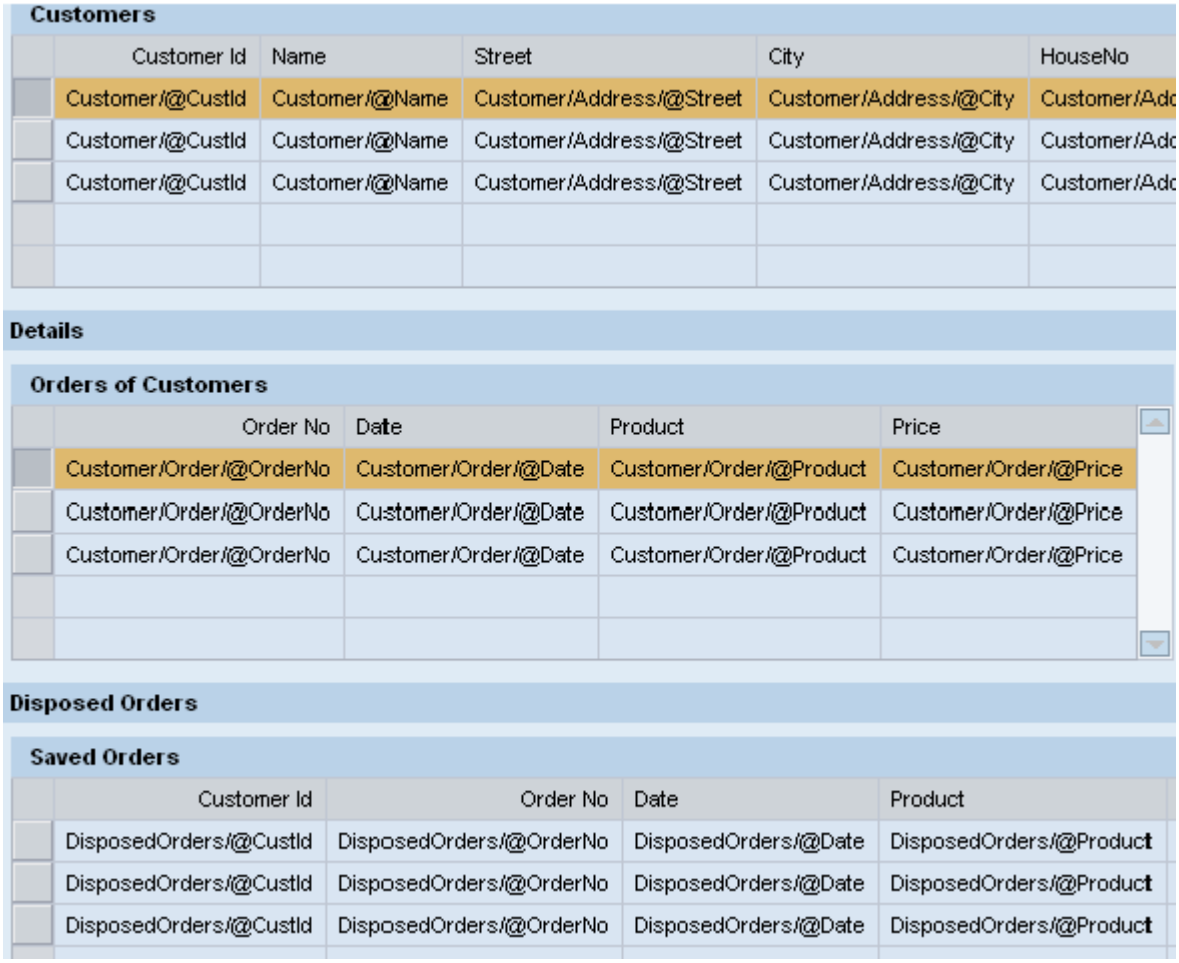


Designing the View Layout

The view layout will have three tables.

- First table is for displaying customer details.
- The second table shows the order details for the selected customer.
- The third table shows the saved orders.
- If the user updates the order information in the second table (Orders of customers), the updated data is saved and shown in the third table (Saved Orders).
- The data source property of each table should be mapped to the respective nodes in the context.

The view layout looks like this



Implementing Supply Function

The supply function is implemented in the component controller. As soon as the singleton property of Order node is set as true, a supply function is added automatically.

The implementation details of wdDoInit() method of component controller is given below. Here SomeBOL class provides the sample data for the Customer and the Order node. Resource bundles, database can also be used to supply data to the context nodes.

In the wdDoInit() method of component controller, the context is populated with the data using the SomeBOL class.

```
public void wdDoInit()
  {
    //@@begin wdDoInit()

        SomeBOL.Customer customer;
        SomeBOL.Address address;
        ICustomerElement newCustomerNodeElement;
        IAddressElement newAddressNodeElement;

        someBOL.initialize();
        Collection customers = someBOL.getCustomers();

        // ======== Populate context ==================
        for (Iterator iter = customers.iterator(); iter.hasNext();){
          customer = (SomeBOL.Customer)iter.next();
          address = customer.getAddress();
          // Instantiate new value node element of type ICustomersElement
          // and define attributes
          newCustomerNodeElement = wdContext.createCustomerElement();
          newCustomerNodeElement.setCustId(customer.getCustId());
          newCustomerNodeElement.setName(customer.getName());
          // The creation of a new inner node element instance requires
          // the existence of a parent node element already bound to the
          // parent node. So call method bind() (or alternatively
addElement()) before.

          wdContext.nodeCustomer().addElement(newCustomerNodeElement);
          // Instantiate new value node element of type IAddressElement
          // and define attributes.

          newAddressNodeElement = wdContext.createAddressElement();
          newAddressNodeElement.setCity(address.getCity());
          newAddressNodeElement.setCountry(address.getCountry());
          newAddressNodeElement.setHouseNo(address.getHouseNo());
          newAddressNodeElement.setPostalCode(address.getPostalCode());
          newAddressNodeElement.setStreet(address.getStreet());

          // the dependent context value node Address can only be referenced
here for each context value node element of type ICustomersElement separately
because we defined it to be a non-singleton node.
          newCustomerNodeElement.nodeAddress().bind(newAddressNodeElement);

        }
    //@@end
  }
```

The code snippet for the supply function, supplyOrder is given below.

```
public void supplyOrder(IPrivateSupplyDisposeComp.IOrderNode node,
IPrivateSupplyDisposeComp.ICustomerElement parentElement)
  {
    //@@begin supplyOrder(IWDNode,IWDNodeElement)
        SomeBOL.Order order;
        IOrderElement newOrderNodeElement;

        // get list of Orders for given customer from some BOL
        Collection orders =
someBOL.getOrdersForCustomer(parentElement.getName());
        for (Iterator iter = orders.iterator(); iter.hasNext();){
          order = (SomeBOL.Order)iter.next();
          // create new instance of a node element of type Order

          newOrderNodeElement = wdContext.createOrderElement();

          // set context value attributes contained in context value node of
type Order

          newOrderNodeElement.setOrderNo(order.getOrderNo());
          newOrderNodeElement.setDate(order.getDate());
          newOrderNodeElement.setPrice(order.getPrice());
          newOrderNodeElement.setProduct(order.getProduct());

          // Add new node element of type Order to node of type
IPrivateWork.IOrderNode.
          // This means: add node element newOrderNodeElement to collection
of
          // node elements hold by singleton context value node node (Order)

          node.addElement(newOrderNodeElement);
        }
    //@@end
  }
```

Implementing Dispose Function

Dispose function is implemented as an inner class within the component controller. The inner class implements the IWDNodeCollectionDisposer interface. The instance of the dispose function inner class is assigned to the appropriate context node.

Add the below two lines in the wdDoInit() method of the component controller

```
DisposeOrders orderDisposer = new DisposeOrders();
wdContext.nodeOrder().getNodeInfo().
            setCollectionDisposer(orderDisposer);
```

Each time the lead selection in the Customer node changes, the dispose function will copy the modified contents of the Order node into the Disposedorders node.When the lead selection changes, LeadSelectionEvent is triggered and the disposed function is executed. Then the updated Order data is copied to the DisposedOrders node.

The implementation of dispose function inner class is given below.

```
private class DisposeOrders implements IWDNodeCollectionDisposer {
          // Inner class for buffering line items after they have been
changed by the user and
          // are no longer required on the screen
          public void disposeElements(IWDNode node) {
            // Cast the incoming node as a LineItems node
            IOrderNode orderNode = (IOrderNode)node;

            // Get a reference to the DisposedOrders node
            IDisposedOrdersNode savedNode = wdContext.nodeDisposedOrders();
            int numOrders = orderNode.size();
            int numDisposed = savedNode.size();

            // Get the currently selected customer
            int currentCustomer =
((ICustomerElement)orderNode.getParentElement()).getCustId();
            boolean alreadySaved = false;

            // Loop around the DisposedOrders node looking for the order
number
            for (int i = 0; i < numDisposed; i++){
              if (savedNode.getDisposedOrdersElementAt(i).getCustId() ==
currentCustomer) {
                alreadySaved = true;
                i = numDisposed;
              }
            }
            if (!alreadySaved)
              for (int j = 0; j < numOrders; j++) {
                IDisposedOrdersElement newEl =
savedNode.createDisposedOrdersElement();

                // Set the OrderNo attribute manually but copy all other
attributes using
                // the WDCopyService

                newEl.setCustId(currentCustomer);

WDCopyService.copyCorresponding((IOrderElement)node.getElementAt(j), newEl);
                savedNode.addElement(newEl);
              }
          } // End method disposeElements()
        } //
  //@@end
}
```

Build, deploy and run the application.

Output

In the Customers table, first customer is selected.

Customers

Customer Id	Name	Street	City	HouseNo
1,002	Smith	Square Garden	New York	215 West 34th Street
1,001	Schmidt	Bahnhofstraße	Berlin	127
1,000	Miller	Tabernacle Street	London	10

The orders placed by the customer are displayed in the Orders table.

Orders of Customers

Order No	Date	Product
10,021	13.10.1997	Floor Lamp
10,022	23.05.2001	Commode
10,023	08.01.2002	Rack
10,024	17.08.1998	Lamp (Halogene)
10,025	11.10.1997	Davenport

The user is updating the product in the orders table

Orders of Customers

Order No	Date	Product
10,021	13.10.1997	Rack
10,022	23.05.2001	Commode
10,023	08.01.2002	Rack
10,024	17.08.1998	Lamp (Halogene)
10,025	11.10.1997	Davenport

Then the user is selecting a second customer in the Customers table. The updated product is saved and displayed in the Saved orders table.

Customers

Customer Id	Name	Street	City	HouseNo
1,002	Smith	Square Garden	New York	215 West 34th Street
1,001	Schmidt	Bahnhofstraße	Berlin	127
1,000	Miller	Tabernacle Street	London	10

Details

Orders of Customers

Order No	Date	Product
10,011	13.10.1999	Table Noire
10,012	23.05.2001	Green Lamp
10,013	04.01.2000	Desk
10,014	17.07.2000	Lamp
10,015	21.07.2002	Wall Closet

Disposed Orders

Saved Orders

Customer Id	Order No	Date	Product
1,002	10,021	13.10.1997	Rack
1,002	10,022	23.05.2001	Commode
1,002	10,023	08.01.2002	Rack
1,002	10,024	17.08.1998	Lamp (Halogene)
1,002	10,025	11.10.1997	Davenport

It is done.

Related Content

https://www.sdn.sap.com/irj/scn/go/portal/prtroot/docs/library/uuid/60086bd5-9301-2b10-6f97-a14366a5602b

For more information, visit the Web Dynpro Java homepage.

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.