Advanced Input Help - The Object Value Selector (OVS)

SAP NetWeaver 04

Copyright

Icons in Body Text

Icon	Meaning
	Caution
	Example
	Note
	Recommendation
	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.

Advanced Input Help - The Object Value Selector (OVS)

Summary

In this chapter you will learn how to embed the most sophisticated generic input help service provided by the Web Dynpro Java Foundation Framework. This input help service is known as the *Object Value Selector (OVS)*. A generic OVS user interface provides all functionality for entering and executing a search query and copying values of a selected object back to one or more input fields.

To use the OVS in your Web Dynpro application, you have to implement some controller code. First, the underlying application context attributes have to be *OVS-enabled* with the Web Dynpro service class `WDValueServices` so that input fields bound to these context attributes can open the OVS UI.

The next step is to declare an *OVS helper context* in a separate OVS custom controller for storing search input and output data.

Finally the `IWDOVSContextNotificationListener` interface has to be implemented as an inner class in the OVS custom controller. This OVS listener class acts as a kind of OVS controller for handling OVS-related UI, such as opening the OVS dialog box, triggering the OVS search query, or selecting a query result.

Keywords

Object Value Selector, OVS, Generic OVS-UI, Generic UI-Services, input help, IWDOVSContextNotificationListener-API, WDValueServices service class, OVS helper context, OVS custom controller, searching business objects

OVS Sample Application

Based on a simple flight search scenario, a Web Dynpro application demonstrates how to use the Object Value Selector input help in Web Dynpro applications (see section Running the Web Dynpro OVS Input Help Sample Application [page 13]).

The flight data is retrieved by connecting to the remote SAP backend system using an Adaptive RFC model.

Content

Next Step:

Introduction [page 6]

Introduction

Different Types of Web Dynpro Input Help

Simple Value Selector

The Web Dynpro Foundation Framework features three types of generic input help UI services. The simplest one is the *Simple Value Selector (SVS)*, which is based on a *DropDownByKey* UI element. The texts to be displayed in the dropdown list are retrieved from a simple data type's enumeration of key/display text pairs. To make use of the generic SVS input help, an application developer just has to declare the data binding between the UI element property *selectedKey* and a context attribute. At runtime, the simple data type of this context attribute must contain a set of key/display text pairs. A simple data type can have different origins. One is the project internal local Java Dictionary. This Dictionary stores all simple and structure types statically declared at design time. Another origin of a simple data type is the *Logical Dictionary* belonging to an imported Adaptive RFC model. Finally, you can modify the data type of a single context attribute programmatically, so that it either becomes a simple data type or it changes its metadata (for example, by adding further key/value pairs to a statically declared simple data type).

More information about the Simple Value Selector input help is available in the tutorial Value Help in Web Dynpro Applications [extern].

Extended Value Selector

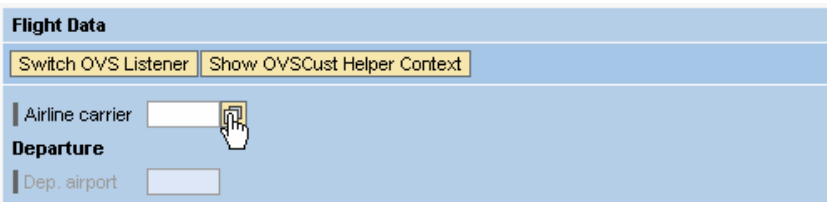
The second Web Dynpro input help is the *Extended Value Selector (EVS)*. Like the SVS, this input help is used for selecting a key/display text pair of a simple data type. Because the SVS is not suitable for displaying large value sets (more than 50, for example), Web Dynpro provides the EVS. This input help displays a popup UI with a built-in function for browsing and filtering large value sets in a table. The EVS can be displayed for every *Inputfield* UI element with the value property bound to a context attribute of the type *simple data type* (at runtime).

More information about the Extended Value Selector input help is available in the tutorial Value Help in Web Dynpro Applications [extern].

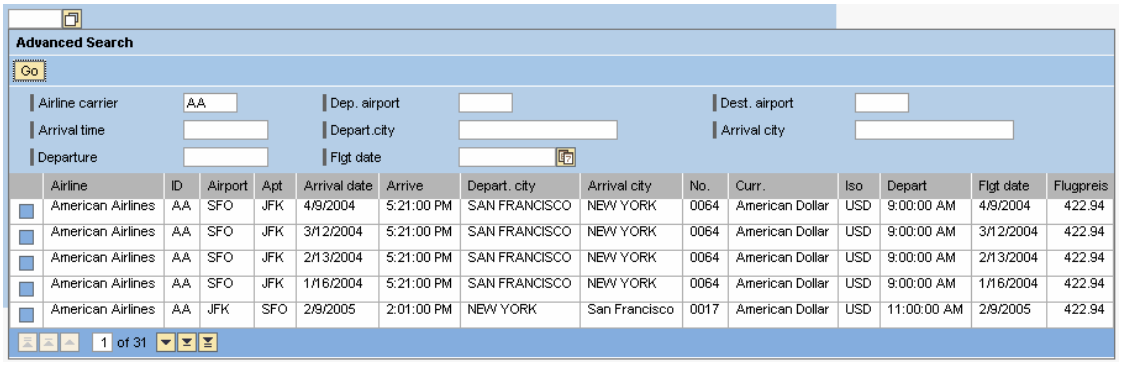
Object Value Selector

In many application scenarios, an additional type of input help is needed to search for objects instead of values. For example, when searching for an *Airline ID*, you can enter some relevant data, such as departure and arrival airports or the flight date, in a search form. The search results (matching flight objects) are displayed in a table and, after selecting a flight, the airplane ID (or other values) is (are) automatically transferred to the corresponding input field(s). For this purpose, Web Dynpro provides a third type of generic input help service called *OVS (Object Value Selector)*. Look at the following screenshots:

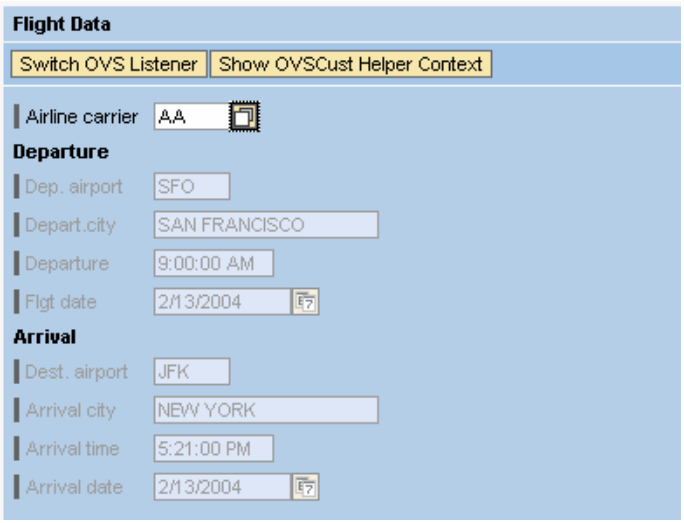
- The OVS UI can be opened by pressing the input help button beneath an OVS-enabled input field



- The search results for the specified search query are displayed in the generic OVS UI



- Data of the table row selected in the OVS UI is displayed in the form input fields



Next Step:

.The Object Value Selector [page 7]

The Object Value Selector

Unlike the SVS and EVS, the Object Value Selector is not entirely based on a declarative approach. To embed this sophisticated input help in your Web Dynpro application, you have to implement some lines of code in a corresponding OVS custom controller. In return for your programming effort, the Web Dynpro runtime automatically renders a generic OVS UI. This user interface is based on a special OVS core component belonging to the Web Dynpro Java runtime environment.

How to Use the OVS Value Help

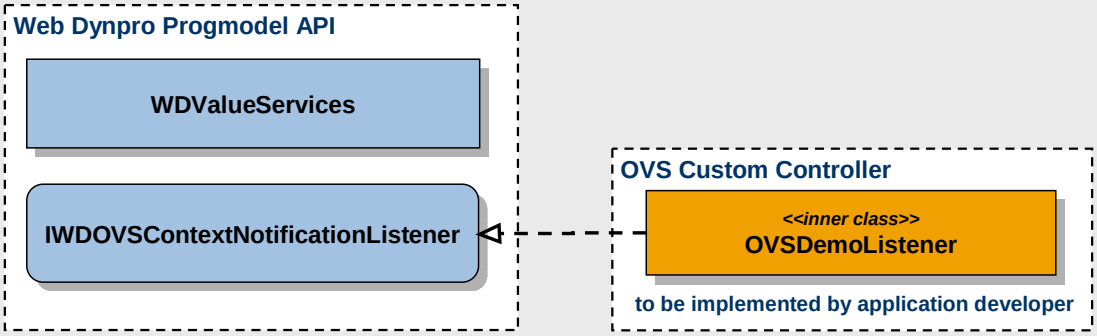
Using the OVS is based mainly on the following three steps:

1. **OVS helper context**: The OVS stores its data (search input data and the query results) in an OVS helper context, which is part of the OVS custom controller. The application developer declares a context node of cardinality 1..1 for storing the search input data and another node of cardinality 0..n for storing the query results. If a suitable RFC function is available, you just have to declare the model binding. The input and output node are then part of the created context structure.
2. **OVS listener**: The OVS uses a listener implementation of the interface `IWDOVSContextNotificatiionListener`. This listener implements three hook methods, which act as event handlers and are called at three separate points in time:

When the OVS UI is requested for a field, when the search query is triggered, and when the user selects a row in the search result table (see figures below).

3. **OVS extension**: The OVS is used for finding objects and copying object values to the context (to context attributes). It is an input help service provided for all context attributes (that is, input fields bound to these context attributes), which were programmatically extended with the OVS feature. The Web Dynpro programming model API provides the `WDValueServices` class for adding this OVS extension to context attributes.

OVS-Related Classes and Interfaces

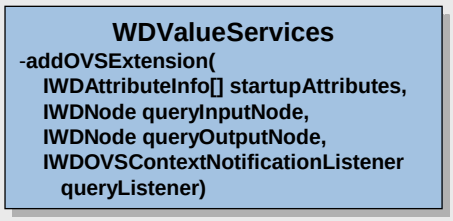


WDValueServices service class

Provides context attributes (`startupAttributes`) with OVS functionality.

Uses separate context nodes for storing search query input values (`queryInputNode`, lead selection points to node element) and search query results (`queryOutputNode`, cardinality 0..n).

Uses a query listener class (`queryListener`), which implements `IWDOVSContextNotification-Listener` for performing the search query and copying context data between the OVS and the application.



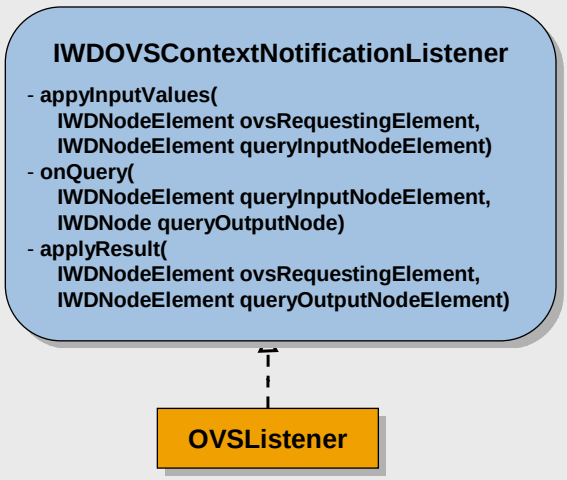
IWDOVSContextNotificationListener API

The OVS listener's hook methods can be seen as event handlers called at three points in time:

`applyInputValues()`: When the OVS is requested for a field. Application context data (`ovsRequestingElement`) is copied to the OVS search query context (`queryInputNodeElement`).

`onQuery()`: Performs search query based on search input (`queryInputNode- Element`). Copies search results in helper context node (`queryOutputNode`).

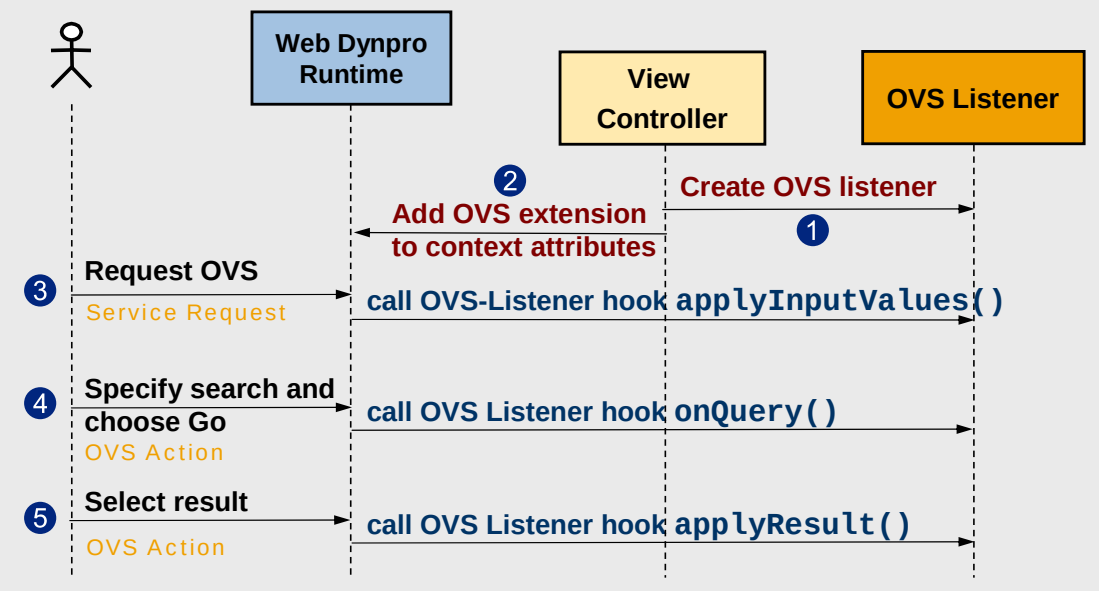
`applyResult()`: User selects a single row in the search result table.Selected OVS query context (`queryOutput-NodeElement`) is copied to the



application context (`ovsRequestingElement`).

How the OVS Input Help Works at Runtime

The runtime aspects of the OVS input help are illustrated in the following sequence diagram:



4. Instantiation of an OVS listener object, which implements the interface `IWDOVSContextNotificationListener`. This is done in the `wdDoInit()` hook method, which is automatically called at controller startup. In this example, the context to be OVS-enabled belongs to a view controller.
5. Adding the OVS extension to context attributes. This is done by calling `WDValueServices.addOVSExtension(...)`. The signature of the method `addOVSExtension()` is displayed in the previous figure. The startup attributes to be OVS-enabled are passed in an array of the type `IWDAttributeInfo`. In addition, references to the OVS helper context nodes (`queryInputNode` and `queryOutputNode`) and to the used OVS listener implementation have to be passed to the Web Dynpro class `WDValueServices`.
6. When a user requests the OVS input help for an input field by clicking the input help button, a Web Dynpro service request is sent to the server. The Web Dynpro runtime then calls the hook `applyInputValues()` implemented by the OVS listener class. This method obtains a reference to the context node element to which the input field is bound to as well as a reference to the context node element that stores the OVS query input data. This OVS hook method initializes the input fields of the OVS search form.
7. When you choose *Go* in the OVS dialog box, an OVS action is sent to the Web Dynpro runtime, which then calls the OVS listener's method `onQuery()`. By accessing the query input data, you can execute a search query to the back end. The retrieved search results can then be stored in the OVS helper context node `queryOutputNode`.
8. The user then selects a row in the OVS search result table. Another OVS action is sent to the Web Dynpro runtime. The third hook method of the OVS listener class `applyResult()` is called for copying values in the selected result object (`queryOutputElement`) to the context node element to which the input field (from where the OVS was opened) is bound.

Next Step:

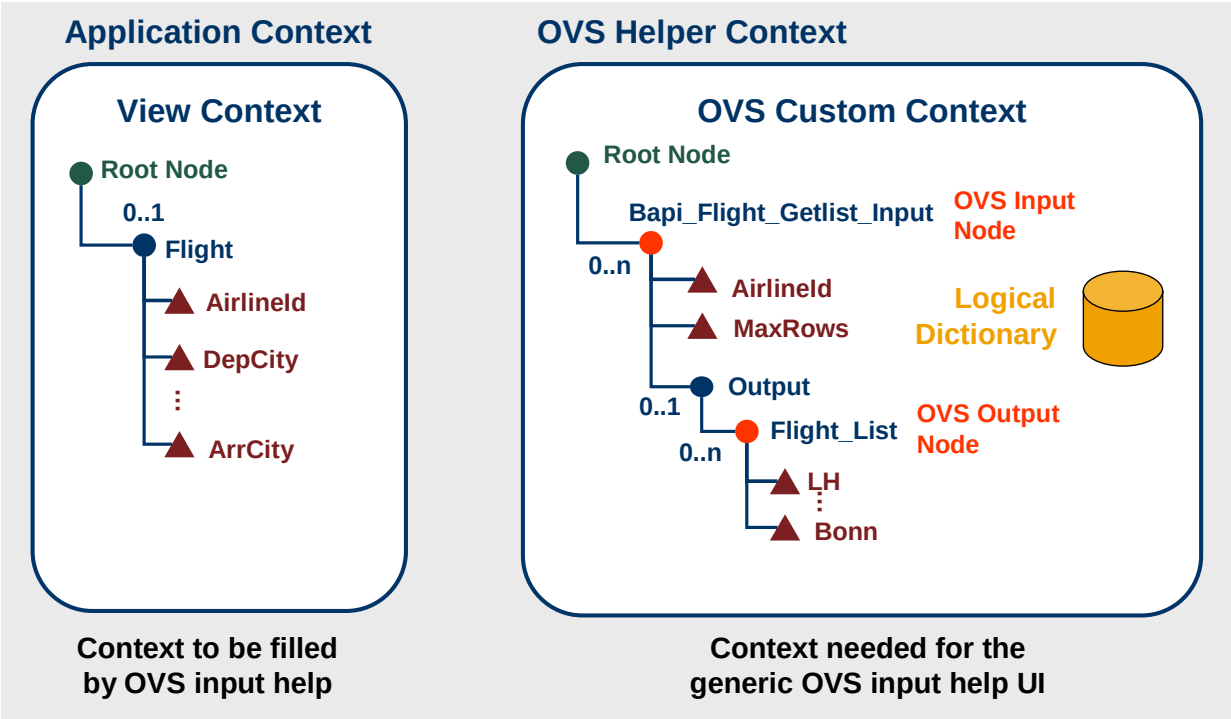
Adding an OVS Extension to the Context [page 10]

Adding an OVS Extension to the Context

The OVS extension of context attributes is specified programmatically. In this example, a view controller context (called *application context*) stores flight data in a value node *Flight* with a number of attributes. In a view layout, some input fields are bound to these attributes. We now want to add the OVS input help function to the *Airline ID* input field. For this reason, the following lines of code have to be implemented in the view controller's `wdDoInit()`method.

```
//@@begin javadoc:wdDoInit()
/** Hook method called to initialize view controller. */
//@@end
public void wdDoInit() {
//@@begin wdDoInit()
  IWDAttributeInfo[] ovsStartUpAttributes = {
    wdContext.nodeFlight().getNodeInfo().getAttribute("Airlineid")};
  IWDOVSContextNotificationListener listener =
    wdThis.wdGetOVSCustController().getOVSListener();
  WDValueServices.addOVSExtension("Flight Selection", // not used yet
    // fields bound to startup attributes will be OVS-enabled
    ovsStartUpAttributes,
    wdThis.wdGetOVSCustController().getOVSInputNode(),
    wdThis.wdGetOVSCustController().getOVSOutputNode(),
    listener);
//@@end
}
```

In this example, the OVS helper context exposes public methods in order to pass references to the OVS listener object and the OVS helper context nodes *Input* and *Output*. These methods have to be declared for the OVS custom controller in the SAP NetWeaver Developer Studio. The OVS helper context contains a structure based on the RFC function BAPI_FLIGHT_GETLIST_INPUT (based on an Adaptive RFC model binding declaration). Therefore, the OVS input node is called *Bapi_Flight_Getlist_Input* and the output node is named *Flight_List* (see next figure).



Next Step:

Implementing the OVS Custom Controller [page 11]

Implementing the OVS Custom Controller

The Generic OVS UI Requires Dictionary Metadata

The *OVS custom controller* stores the OVS helper context data, which is requried by the generic OVS UI. It is important to note that the context attributes in this helper context must be appropriately typed with simple data types in a local or logical Data Dictionary. Metadata of these simple types is read by the generic OVS core component for dynamically creating the OVS user interface with field labels, tool tips or column header texts. Without access to this metadata, the generic OVS UI cannot be correctly rendered.

Declaring the OVS Context Nodes

In some application scenarios, the back end does not provide an *OVS-enabled* RFC function. OVS-enabled means that all required input attributes are on the same level in one input structure (OVS input node after model binding). Likewise, all attributes to be displayed in the search result table have to belong to one output structure (OVS output node after model binding). Thus, the OVS does not support input or output attributes on different levels. The way round this restriction is to declare separate OVS input or output nodes. These nodes are value nodes with all required attributes, properly typed with simple types of the Logical Adaptive RFC Dictionary. The copying of data between the used model nodes and the additional OVS input and/or output value nodes has to be implemented in the OVS listener.

Binding an Executable Model Object to the OVS Input Node

The OVS custom controller contains the implementation of the required `IWDOVSContextNotificationListener` interface as an inner class (see next step). Because the OVS requires a reference to an existing input node element, we have to bind a model object of the type *Bapi_Flight_Getlist_Input* to the identically-named context model node (OVS input node) at controller startup:

```
//@@begin javadoc:wdDoInit()
/** Hook method called to initialize OVS custom controller. */
//@@end
public void wdDoInit() {
//@@begin wdDoInit()
  Bapi_Flight_Getlist_Input bapiInput = new Bapi_Flight_Getlist_Input();
  wdContext.nodeBapi_Flight_Getlist_Input().bind(bapiInput);
//@@end
}
```

Implementing Public Methods of the OVS Custom Controller

Accessor methods within the OVS custom controller's public API expose the OVS listener object as well as the OVS input and output nodes to other controllers.

```
//@@begin javadoc:getOVSListener()
/** Declared method. */
//@@end
public
com.sap.tc.webdynpro.progmodel.api.IWDOVSContextNotificationListener
getOVSListener( ) {
  //@@begin getOVSListener()
  return ovsListener;
  //@@end
}
```

```
//@@begin javadoc:getOVSInputNode()
/** Declared method. Called by the OVS view controller. */
//@@end
public com.sap.tc.webdynpro.progmodel.api.IWDNode getOVSInputNode( ) {
  //@@begin getOVSInputNode()
  return wdContext.nodeBapi_Flight_Getlist_Input();
  //@@end
}

//@@begin javadoc:getOVSOutputNode()
/** Declared method. */
//@@end
public com.sap.tc.webdynpro.progmodel.api.IWDNode getOVSOutputNode( ) {
  //@@begin getOVSOutputNode()
  return wdContext.nodeFlight_List();
  //@@end
}
```

Next Step

Implementing the Interface IWDOVSContextNotificationListener [page 12]

Implementing the Interface IWDOVSContextNotificationListener

The implementation of the interface `IWDOVSContextNotificationListener` is contained in the OVS custom controller as an inner class. The OVS custom controller stores an instance of this listener in a private member variable: `private IWDOVSContextNotificationListener ovsListener = new OVSDemoContextNotificationListener();` This object is returned by the public controller method `getOVSListener()`. To enable typed access to the OVS helper context, the corresponding type casts are required.

```
...
//@@begin others
/* Implement IWDOVSContextNotificationListener as inner class of
 * custom controller OVSCust. */
private class OVSDemoContextNotificationListener
  implements IWDOVSContextNotificationListener {
  /* @see com.sap.tc.webdynpro.progmodel.api
   *         .IWDOVSContextNotificationListener#onQuery(...) */
  public void onQuery(IWDNodeElement queryInputNodeElement,
                      IWDNode queryOutputNode) {
    IPublicOVSCust.IBapi_Flight_Getlist_InputElement ovsInput =
      (IPublicOVSCust.IBapi_Flight_Getlist_InputElement)
        queryInputNodeElement;
    IPublicOVSCust.IFlight_ListNode ovsOutput = (
      IPublicOVSCust.IFlight_ListNode) queryOutputNode;
    try {
      ovsInput.modelObject().execute();
      // invalidate 'Output' model node via a top-down access
      // approach
      ovsInput.node().getChildNode("Output",0).invalidate();
```

```
    } catch (Exception e) {
      IWDMessageManager msgMgr = wdComponentAPI.getMessageManager();
      msgMgr.reportException(e.getLocalizedMessage(), false);
    }
  }

  /* @see com.sap.tc.webdynpro.progmodel.api
   *         .IWDOVSContextNotificationListener#applyResult(...)*/
  public void applyResult(IWDNodeElement applicationNodeElement,
                          IWDNodeElement queryOutputNodeElement) {
    IPrivateOVSView.IFlightDataElement ovsCallerNodeElement =
     (IPrivateOVSView.IFlightDataElement) applicationNodeElement;
    IPublicOVSCust.IFlight_ListElement output =
     (IPublicOVSCust.IFlight_ListElement) queryOutputNodeElement;
    ovsCallerNodeElement.setAirlineid(output.getAirlineid());
    ...
    ovsCallerNodeElement.setDeptime(output.getDeptime());
  }

  /* @see com.sap.tc.webdynpro.progmodel.api
   * .IWDOVSContextNotificationListener#applyInputValues(...) */
  public void applyInputValues(IWDNodeElement applicationNodeElement,
                               IWDNodeElement queryInputNodeElement) {
    Object initialValue = applicationNodeElement
                            .getAttributeValue("Airlineid");
    queryInputNodeElement.setAttributeValue("Airline", initialValue);
  }
}

private IWDOVSContextNotificationListener ovsListener =
          new OVSDemoContextNotificationListener();
//@@end
```

Conclusion

The Web Dynpro Object Value Selector simplifies the embedding of a user interface for finding objects in your Web Dynpro application. Whereas the SVS and the EVS input helps are used for finding key and display texts contained in simple data types, the Object Value Selector allows you to search for objects. The OVS user interface is automatically generated by the Web Dynpro runtime. It is available to all context attributes with a programmatically specified OVS extension using the Web Dynpro service `WDValueServices`. An OVS helper context is used to store the OVS query input and output data. The contained helper context elements have to be typed using simple Dictionary types. The OVS helper context and the OVS listener class implementing the `IWDOVSContextNotificationListener` interface are part of a separate OVS custom controller

Web Dynpro OVS Sample Application

Running the Web Dynpro OVS Input Help Sample Application [page 13]

Running the Web Dynpro OVS Input Help Sample Application

This article is based on a Web Dynpro sample application, which is available on the *SAP Developer Network (SDN)* at http://sdn.sap.com. The project is available for download in a

corresponding zip file under the category *Home → Developer Areas → Web Application Server → Web Dynpro → Sample Applications and Tutorials.*

Prerequisites

- The sample application is based on a *SAP NetWeaver 04 – Stack 11* installation

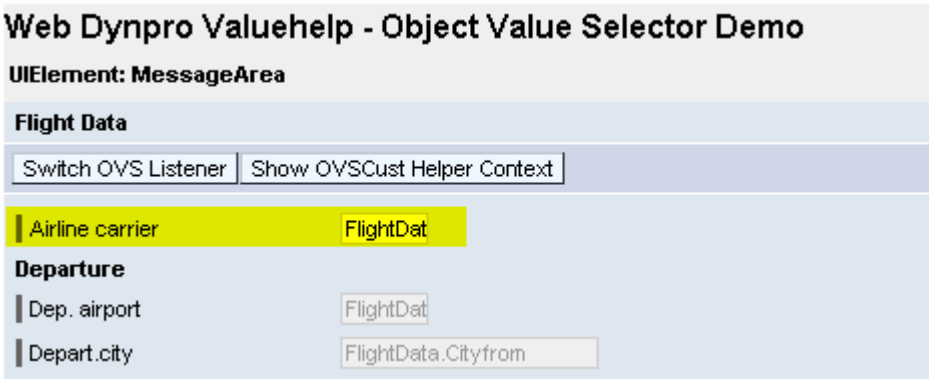
  The application calls the *Bapi_Flight_Getlist_Input* RFC function module for retrieving flight data displayed in the UI. To run this sample project, follow the steps described in the **readme** file of the OVS input help sample application in SDN.

Web Dynpro Project Structure

The sample application comprises the following project structure:

- **Web Dynpro Project name**: `WebDynpro_OVS_Valuehelp`
- **Web Dynpro Application**: `OVSDemoApp`
- **Web Dynpro Component**: `OVSDemoComp`
  - **Custom Controller**: `OVSCust`
    Implements the OVS listener classes
    - **Public methods**: *getOVSListener(), switchOVSListern(), getOVSInputNode(), getOVSOutputNode().*
    - **Inner classes:** *OVSDemoContextNotificationListener, AnotherOVSDemoContextNotificationListener*
      Implement the *IWDContextNotificationListener-API*
    - **Member variables:** *tempOVSListener, ovsListener, anotherOVSListener*
      Reference listener objects of the type *IWDContextNotificationListener* implemented by inner classes.
    - **Context**:
      - *Bapi_Flight_Getlist_Input*: Model node for executing a flight query.
      - *OVSFlightQueryOutput:* Model node for OVS output for referencing lists of returned flight model objects.
      - *OVSFlightQueryInput*: Value node for storing a custom query input, which is not provided by the given model. Because OVS input attributes must belong to a single node, a value node containing all desired attributes must be declared.
  - **Web Dynpro View**: `OVSView`
    - **Controller Usage**: A controller usage relation to the OVS custom controller is defined for calling its public interface methods *getOVSListener(), switchOVSListern(), getOVSInputNode()*, and *getOVSOutputNode().*
    - **View Layout:**
      The view layout contains a form with an input field for entering an *Airline carrier id*. The corresponding context attribute *FlightData.Airlineid* programmatically receives an OVS extension based on custom controller code. At runtime, the user can then open the generic OVS-UI for this

input field.


Web Dynpro Valuehelp - Object Value Selector Demo
UIElement: MessageArea
Flight Data
Switch OVS Listener
Show OVSCust Helper Context
Airline carrier
FlightDat
Departure
Dep. airport
FlightDat
Depart.city
FlightData.Cityfrom