How to Use Input Field Suggestions in a Web Dynpro for Java Application

Applies to:

Web Dynpro for Java 7.11. For more information, visit the Web Dynpro Java homepage.

Summary

This tutorial explains the input suggestion feature available with the Web Dynpro AJAX client. It is shown how automatic input suggestion can be activated and how an application can implement a customized input suggestion as an extension of the Object Value Selector (OVS) feature.

Author: Web Dynpro Java Team

Company: SAP AG

Created on: 29 June 2010

Table of Contents

Introduction

In this tutorial you will learn how to implement (asynchronous) **input suggestion** for Web Dynpro input fields. Input suggestion displays a list of suggested input values while a user types into the input field.

The input suggestion function in Web Dynpro exists in different forms:

For input fields that are bound to context attributes with value sets, the framework provides **automatic** input suggestion. It is sufficient to just set the “suggestValues” property for the input field. (It does not matter if the value set of the data type has been defined at design time or at runtime.)

Application-controlled input suggestion can be implemented as an extension of the well-known Object Value Selector (OVS) function. The application has full control of the suggestion list in that case.

This tutorial describes both automatic and application-controlled input suggestion. You will learn all the steps for implementing a custom-configured OVS with input suggestion.

To simplify matters, the tutorial application does not use a real data model. Instead, a plain Java class that manages an extract of the geographical database “GeoNames” is used.

The “GeoNames” database is available at http://www.geonames.org/ under a Creative Commons License.

Prerequisites

Systems, Installed Applications, and Authorizations

You need the NetWeaver Developer Studio (Version 7.11 or later) to compile and deploy the tutorial application. The application server used should have the same version as the NWDS or a newer version.

The tutorial application is available as a development component (DC). You need to import the Software Component **HM-WDUIDMKTCNT**, which contains the DC **tc/wd/tut/inpfld/sgst**. The exact steps are described in a separate document.

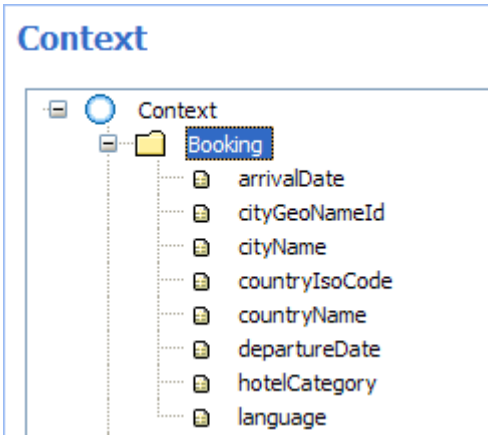
Objectives

After working through this tutorial you should be able to:

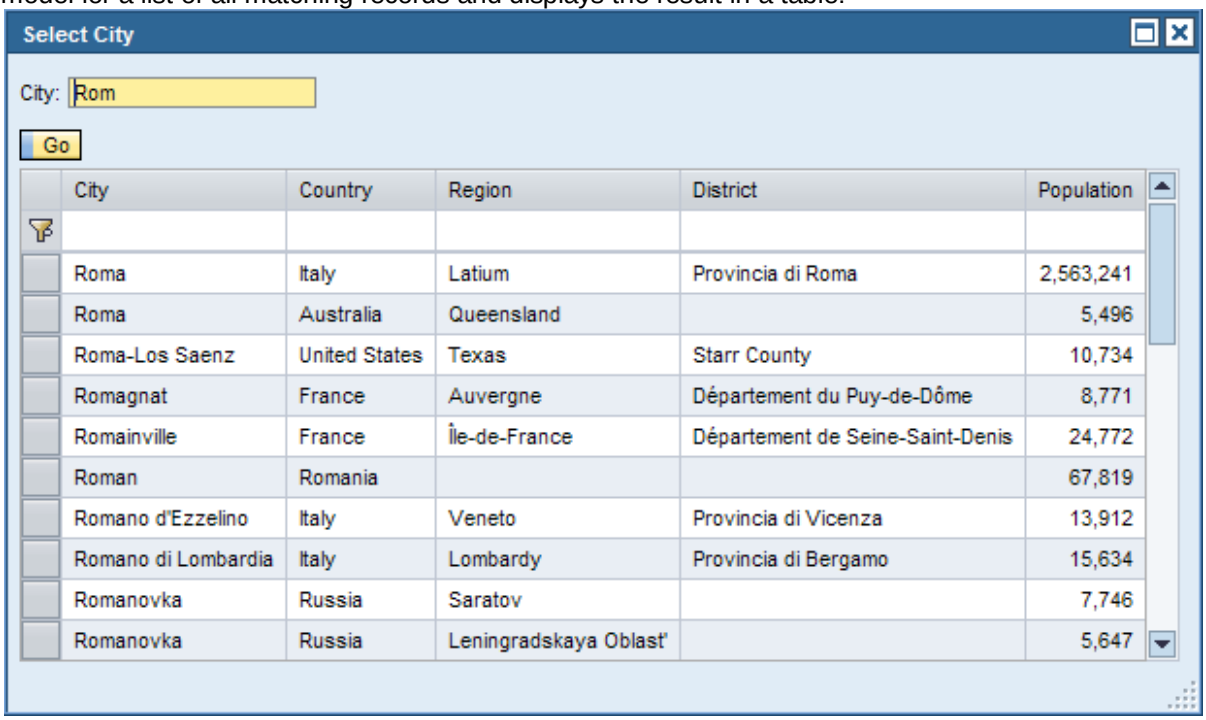
- Apply the input suggestion functionality to your applications
- Implement the core OVS functionality
- Implement input suggestion as an OVS extension
- Implement a custom OVS configuration
- Assign the OVS to a (set of) context attribute(s)
- Activate input suggestion for input fields

An Object Value Selector (OVS) for Cities

The component controller context of our tutorial application contains a node named “Booking” that stores the data for a hotel search form:



We want to create an OVS for the context attribute “cityName“. The OVS dialog allows the user to enter a pattern (in our tutorial, this is just a prefix of the searched city). After you have pressed the “Go” button (this might have a different label, depending on your locale), the OVS queries the data model for a list of all matching records and displays the result in a table:



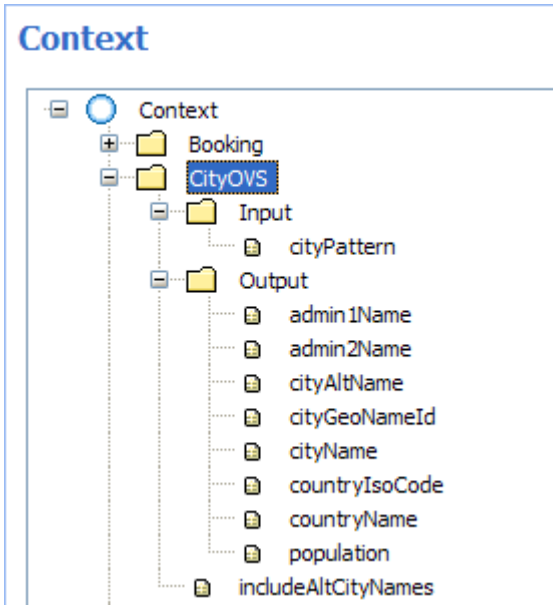
City	Country	Region	District	Population
Roma	Italy	Latium	Provincia di Roma	2,563,241
Roma	Australia	Queensland		5,496
Roma-Los Saenz	United States	Texas	Starr County	10,734
Romagnat	France	Auvergne	Département du Puy-de-Dôme	8,771
Romainville	France	Île-de-France	Département de Seine-Saint-Denis	24,772
Roman	Romania			67,819
Romano d'Ezzelino	Italy	Veneto	Provincia di Vicenza	13,912
Romano di Lombardia	Italy	Lombardy	Provincia di Bergamo	15,634
Romanovka	Russia	Saratov		7,746
Romanovka	Russia	Leningradskaya Oblast'		5,647

The table displays a number of columns that help to identify a city, especially if the city name is not unique. Note that, for example, “Roma” exists twice and can be identified by the country, or that “Romanovka” exists twice and can be identified by the region.

When an entry is selected from the table, the OVS updates some context attributes (city name, city ID, country name, country ID) of the Web Dynpro application and the OVS dialog is closed by the framework. It is up to the OVS implementation to determine which attributes of the application will be updated.

OVS Input and Output Nodes

The OVS needs context nodes to store its input and output values. We use a context node "CityOVS" (cardinality 1:1, selection 1:1) with two sub-nodes "Input" (cardinality 1:1, selection 1:1) and "Output" (cardinality 0:n, selection 0:1) in the component controller context. The elements of the "Output" node represent the rows of the result table;, therefore we give it cardinality 0:n.



Note that only a subset of the output attributes is displayed in the OVS and that the attributes appear in a specific order. We will describe below how this can be achieved.

Implementing the OVS

The component controller contains an inner class "CityOVS**",** which implements the **IWDOVSContextNotificationListener** interface. We implement it as a non-static inner class because we need access to the component controller context and some of its non-static methods.

Let's have a look at the code. Open the source code for the component controller of the tutorial (*Web Dynpro Explorer* → Right-click *Component Controller* → *Open* → *Java Editor*, select *CityOVS* in the *Outline* view):

```
private class CityOVS
  extends WDOVSConfigurator
  implements IWDOVSContextNotificationListener, IWDOVSSuggester
```

Note that this class takes three roles at the same time: OVS context listener, OVS suggester, and OVS configurator.

Implementing the Context Notification Interface

For the core OVS functionality, we implement the **IWDOVSContextNotificationListener** interface methods `onQuery()` and `applyResult()`. The `applyInputValues()` method is not used in our example.

The `onQuery()` method is called by the Web Dynpro runtime when the “Go” button in the OVS dialog is pressed. Its purpose is to query the data model for all records matching the (set of) input parameters and to fill the result table (output node) accordingly.

In our implementation, the data model is queried to find all records where the city name starts with the entered pattern. The parameter “includeAltCityNames” controls whether or not the model should also look for alternative city names.

For each matching record a context element in the output node is created and its attribute values are populated:

```
public void onQuery(IWDNodeElement queryInputNodeElement,
                    IWDNode queryOutputNode)
{
 String filter = queryInputNodeElement.
                 getAttributeAsText(IInputElement.CITY_PATTERN);
 List<GeoRecord> result = getModel().findMatchingRecords(filter,
   wdContext.currentCityOVSElement().getIncludeAltCityNames());
 for (GeoRecord r : result)
 {
  IWDNodeElement e = queryOutputNode.createAndAddElement();
  e.setAttributeValue(IOutputElement.CITY_GEO_NAME_ID, r.getGeoNameId());
  e.setAttributeValue(IOutputElement.CITY_NAME, r.getName());
  if (wdContext.currentCityOVSElement().getIncludeAltCityNames())
  {
   e.setAttributeValue(IOutputElement.CITY_ALT_NAME, r.getMatchingCityName());
  }
  e.setAttributeValue(IOutputElement.COUNTRY_ISO_CODE, r.getCountryCode());
  e.setAttributeValue(IOutputElement.COUNTRY_NAME, getModel().
                      getCountryName(r.getCountryCode()));
  String admin1Key = r.getCountryCode() + "." + r.getAdmin1Code();
  e.setAttributeValue(IOutputElement.ADMIN1_NAME,getModel().
                      getAdmin1Name(admin1Key));
  String admin2Key = admin1Key + "." + r.getAdmin2Code();
  e.setAttributeValue(IOutputElement.ADMIN2_NAME, getModel().
                      getAdmin2Name(admin2Key));
  e.setAttributeValue(IOutputElement.POPULATION, r.getPopulation());
 }
}
```

The `applyResult()` method is called by the Web Dynpro runtime when a row from the output table is selected. The dialog is closed automatically and a number of attribute values in the application context are populated from the selected row.

In our implementation we update the city ID, the city name, the country code, and the country name attributes:

```
public void applyResult(IWDNodeElement applicationNodeElement, IWDNodeElement
                        queryOutputNodeElement)
 {
  String cityGeoNameId = (String)
  queryOutputNodeElement.getAttributeValue(IOutputElement.CITY_GEO_NAME_ID);
  applicationNodeElement.setAttributeValue(IBookingElement.CITY_GEO_NAME_ID,
  cityGeoNameId);
  String cityName = getModel().findRecord(cityGeoNameId).getName();
  applicationNodeElement.setAttributeValue(IBookingElement.CITY_NAME, cityName);
  String countryCode = (String)
  queryOutputNodeElement.getAttributeValue(IOutputElement.COUNTRY_ISO_CODE);
  applicationNodeElement.setAttributeValue(IBookingElement.COUNTRY_ISO_CODE,
  countryCode);
  String countryName = getModel().getCountryName(countryCode);
  applicationNodeElement.setAttributeValue(IBookingElement.COUNTRY_NAME,
  countryName);
 }
```

Note that if you want to use the typed API for the `applicationNodeElement`, you must cast it in accordance with the view controller type and not the component controller type.

That's all that is needed to implement the core OVS functionality!

Implementing the Suggestion Interface

Let's enhance the OVS with the input suggestion functionality. We have to implement the methods from the **IWDOVSSuggester** interface.

The `suggest()` method is called by the Web Dynpro runtime whenever the user types inside a suggestion-enabled input field (after a small delay). This happens asynchronously to avoid blocking the user.

The purpose of the `suggest()` method is to provide a list of input suggestions for the currently entered value (given by the parameter "filter").

A suggestion list entry has three components:

1. A unique key displayed in the first column of the list
2. A description text displayed in the second column
3. An internal key that can be used to identify the data object corresponding to a suggestion

In our implementation, we first query the data model for all city records that match the last value entered in the input field. The API for adding a suggestion list entry is the following method:

```
void addSuggestion(String key, String description, String identifier);
```

The suggestion key is composed from the city name found, the country code, and the two top-most administrative levels to make it unique. (This might not even be sufficient, but as a last resort you can, for example, append a counter to avoid duplicate entries).

As `description` we use the city name together with the country code.

The `identifier` parameter should be a string that can be used to uniquely identify the suggestion value in the `finalize()` method. In our implementation, we use the (unique) key of a city record as identifier:

```
public void suggest(IWDOVSControl ovsControl, String filter, IWDSuggestionList
                    suggestions)
 {
   List<GeoRecord> result = getModel().findMatchingRecords(filter,
     wdContext.currentCityOVSElement().getIncludeAltCityNames());
   for (GeoRecord r : result)
   {
     String suggestionKey = createSuggestionKey(r);
     String countryName = getModel().getCountryName(r.getCountryCode());
     String description = r.getMatchingCityName() + " (" + countryName + ")";
     suggestions.addSuggestion(suggestionKey, description, r.getGeoNameId());
   }
 }
```

The `finalize ()` method is called by the Web Dynpro runtime to store the suggested value back into the context attribute(s) of the application.

In our implementation, the identifier contains the key of the database record for the matching city, and we store the city name, country code, and country name:

```
public void finalize(IWDOVSControl ovsControl, String identifier)
{
  IBookingElement booking = wdContext.currentBookingElement();
  GeoRecord r = getModel().findRecord(identifier);
  if (r != null)
  {
    booking.setCityName(r.getName());
    booking.setCountryIsoCode(r.getCountryCode());
    booking.setCountryName(getModel().getCountryName(r.getCountryCode()));
  }
  else
  {
    booking.setCityName(null);
    booking.setCountryIsoCode(null);
    booking.setCountryName(null);
  }
}
```

Custom OVS Configuration

The appearance of the OVS dialog can be controlled by implementing a subclass of **WDOVSConfigurator** and using an instance of this subclass in the `addOVSExtension()` call.

The OVS configurator class can modify the labels, order, and visibility of the OVS input fields as well as the result table columns. Additionally, the title of the OVS dialog can be modified.

In our implementation, the "CityOVS" class **itself** serves as configurator by extending the class WDOVSConfigurator and overriding its methods.

We override methods `getColumnLabel()` and `getFieldLabel()` to assign custom texts to the input field labels and to the headers of the result table columns. The texts are stored in the component's message pool and accessed through the `IWDTextAccessor` interface:

```
@Override
public String getColumnLabel(String fieldName)
{
  return getFieldLabel(fieldName);
}

@Override
public String getFieldLabel(String fieldName)
{
  return wdComponentAPI.getTextAccessor().getText("city_ovs_field_" + fieldName);
}
```

The dialog title is changed by overriding method `getWindowTitle()`:

```
@Override
public String getWindowTitle()
{
  return wdComponentAPI.getTextAccessor().getText(IMessageTutorial.CITY_OVS_TITLE);
}
```

We want to display the following columns in the output table of the OVS:

City	Country	Region	District	Population

When the "includeAltCityNames" option is enabled, we want to display an additional column:

City	Alternate Name	Country	Region	District	Population

This is achieved by overriding the following method:

```
@Override
public List<String> selectResultColumns(List<String> names)
{
  if (wdContext.currentCityOVSElement().getIncludeAltCityNames())
  {
    return Arrays.asList(
      IOutputElement.CITY_NAME,
      IOutputElement.CITY_ALT_NAME,
      IOutputElement.COUNTRY_NAME,
      IOutputElement.ADMIN1_NAME,
      IOutputElement.ADMIN2_NAME,
      IOutputElement.POPULATION);
  }
  else
  {
```

```
        return Arrays.asList(
          IOutputElement.CITY_NAME,
          IOutputElement.COUNTRY_NAME,
          IOutputElement.ADMIN1_NAME,
          IOutputElement.ADMIN2_NAME,
          IOutputElement.POPULATION);
      }
    }
```

This method should return a list of the context attribute names in the order that is to be used for the table columns. We recommend using the constants generated for the attribute names.

Attaching the OVS to the Context

In the `wdDoInit()` method of the component controller, we assign the OVS to the context attribute that stores the city name:

```
WDValueServices.addOVSExtension
(
  "CityOVS",
  new IWDAttributeInfo[] {wdContext.nodeBooking().getNodeInfo().
                          getAttribute(IBookingElement.CITY_NAME)},
  getCityOVS().getProvider(), /* OVS dialog provider */
  getCityOVS() /* suggester */
);
```

We have encapsulated the creation of and access to the OVS dialog provider inside our `CityOVS` class, but this is not mandatory.

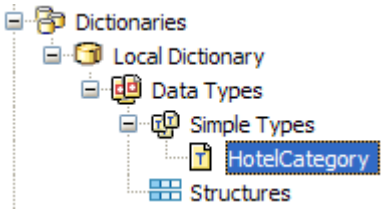
Framework-Controlled Suggestion Support

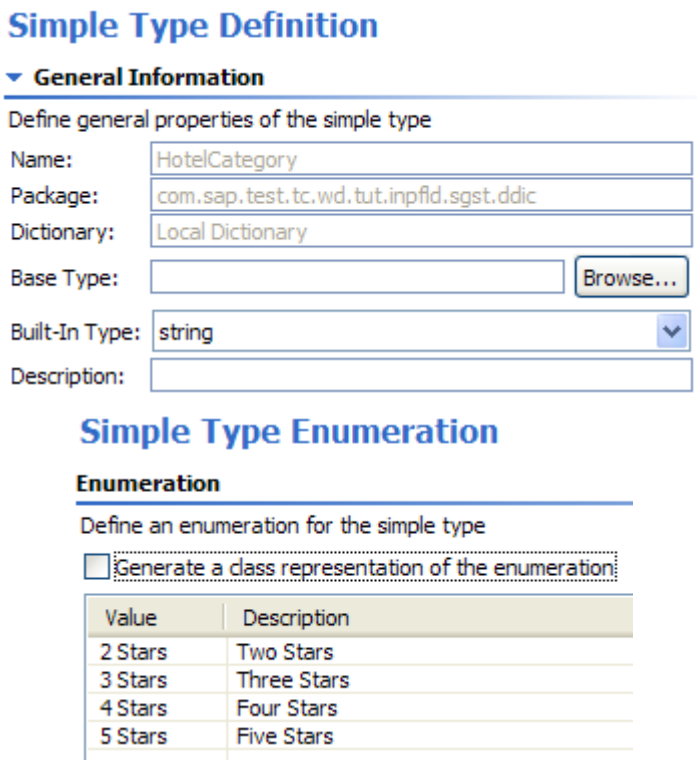
As mentioned above, an **automatic** input suggestion also exists, where the suggestion list is created by the runtime from the value set attached to some DDIC type.

Suggestion with Static Value Sets

In our search form, we have an input field where the user can select his or her preferred hotel category. (This is more for demonstration purposes; a real application would probably just use a drop-down list instead).

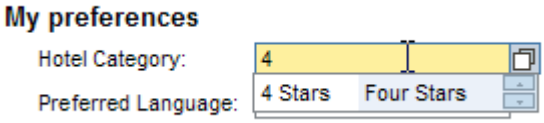
The hotel category values are defined in the DDIC type `HotelCategory`. The value set was created at design time using the DDIC editor:





To activate input suggestion for the corresponding input field, it is sufficient to set the `suggestValues` property to `true`.

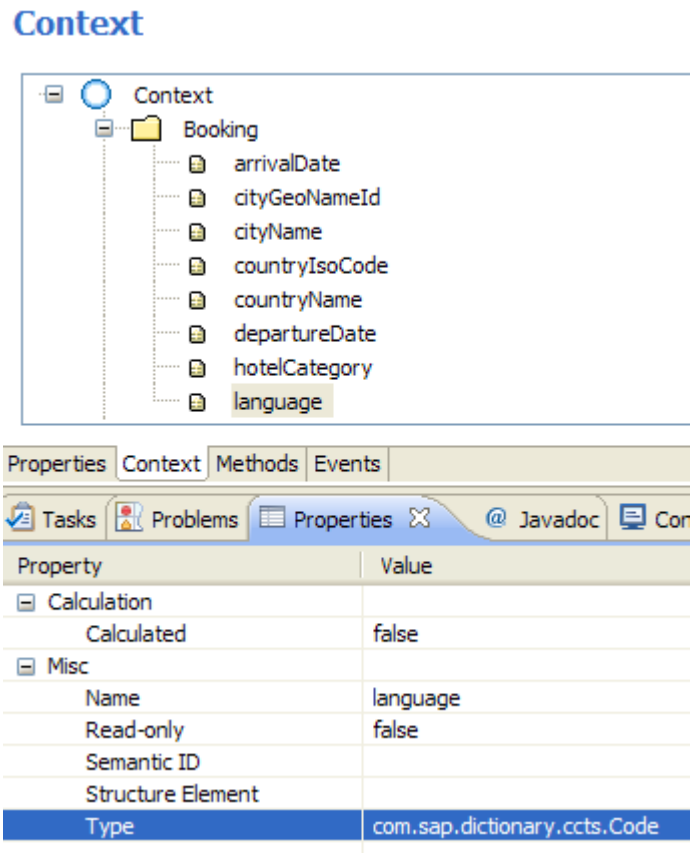
At runtime you will get a suggestion list like the following:



A Dynamic Value Set with Keys of Type CctCode

The input field for entering the preferred language demonstrates the input suggestion together with one of the so called CCTS data types (CCTS = Core Component Technical Specification). We don't want to go into the details of CCTS support in Web Dynpro here. For more information, refer to this SDN article.

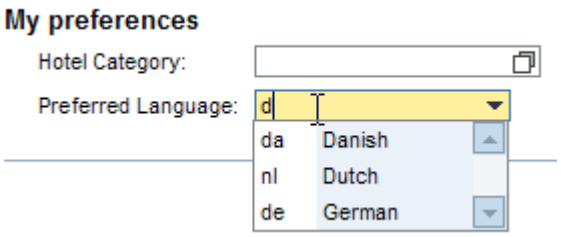
The context attribute `language` is not of type `string` but of CCTS type `Code`:



We create a value set for this attribute, where the keys have the runtime type `CctCode` that corresponds to the design-time type `Code`:

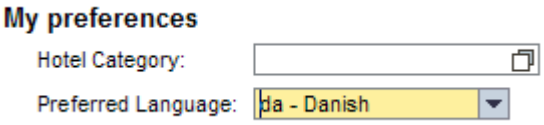
```
private void addLanguages()
{
  IModifiableSimpleValueSet<CctCode> valueSet =
    wdContext.nodeBooking().getNodeInfo()
    .getAttribute(IBookingElement.LANGUAGE).getModifiableScalarType("")
    .getSVServices().getModifiableSimpleValueSet();

  Locale[] locales = Locale.getAvailableLocales();
  Locale sessionLocale = WDResourceHandler.getCurrentSessionLocale();
  Arrays.sort(locales, new CompareLocalesByDisplayName(sessionLocale));
  for (Locale locale : locales)
  {
    CctCode key = new CctCode(locale.getLanguage(), null, null, null, null, null);
    valueSet.put(key, locale.getDisplayLanguage(sessionLocale), sessionLocale);
  }
}
```

Note how the input field is rendered at runtime and how the suggestion feature works here:



When you type inside this field, all languages matching the entered value are offered as suggestions. A language matches the entered value if either the language code or the language name starts with the entered value.

When you select a suggested value and leave the field, the language key and the language name will be displayed inside the field:



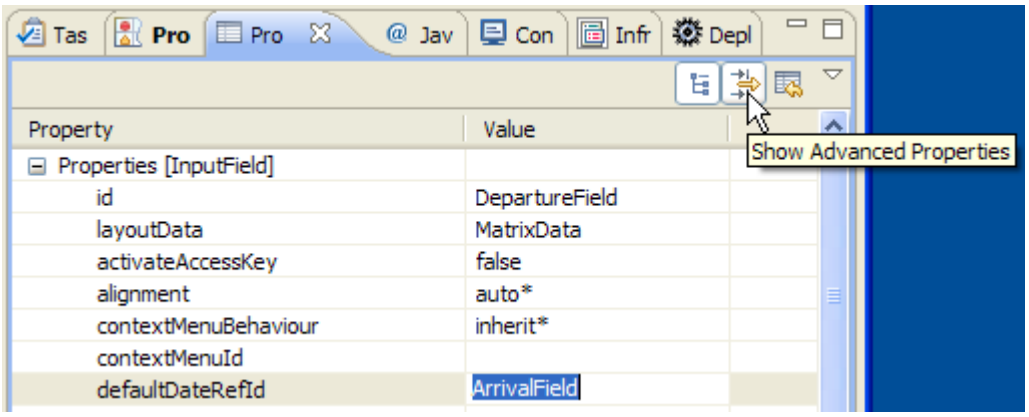
Dependent Date Pickers

Just as an aside, we want to demonstrate a feature (not related to suggestion) that is useful when entering date ranges using date pickers.

The “Arrival Date” and “Departure Date” fields both have a date picker attached as they are bound to context attributes of type `date`.

Select some date in a future month or year for the arrival date. Then open the date picker for the departure field. Note that the date picker starts at the same year and month as the arrival date just entered.

This dependency between date pickers is achieved by setting the `defaultDateRefId` property of the departure field to the ID of the arrival field:



You have to activate the advanced properties in your IDE to see this property.

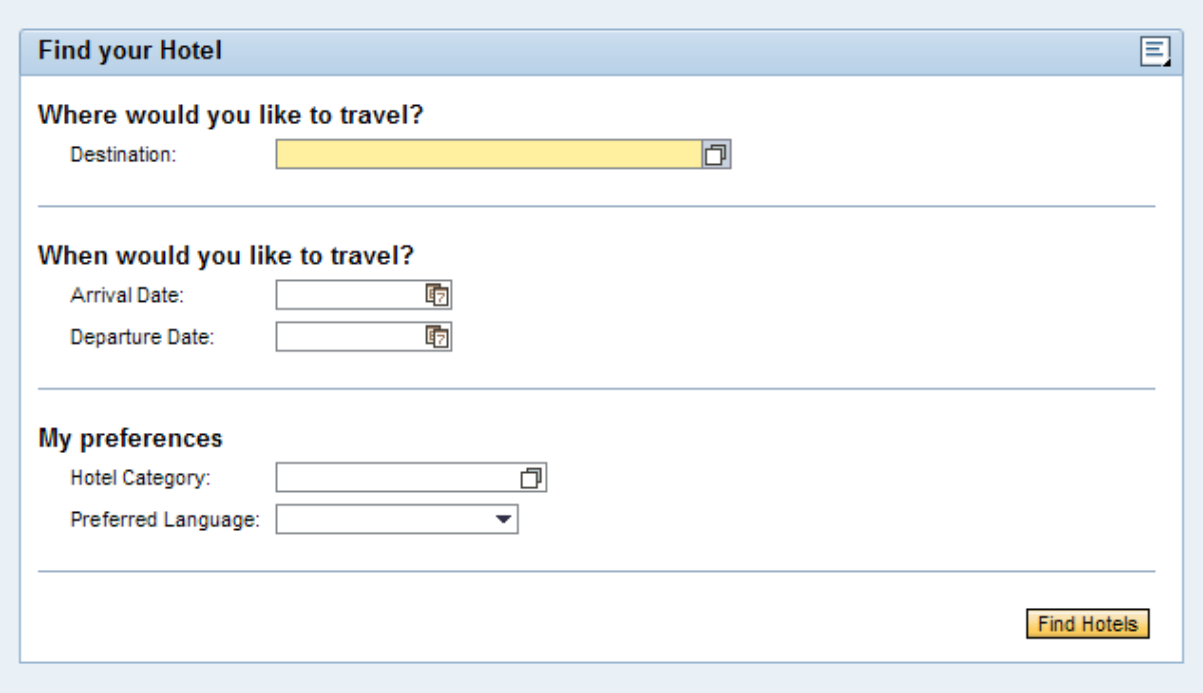
If the dependent field is still empty and the reference field already contains a date, the date picker automatically opens at that date.

Tutorial Result

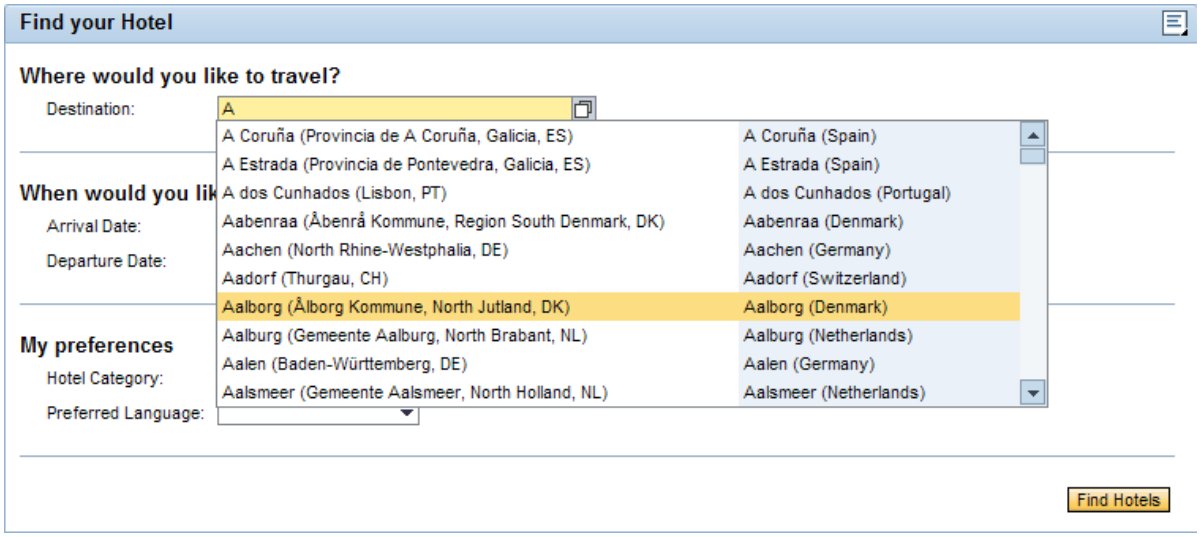
Let's review the features shown in this tutorial.

City Search with OVS and Input Suggestion

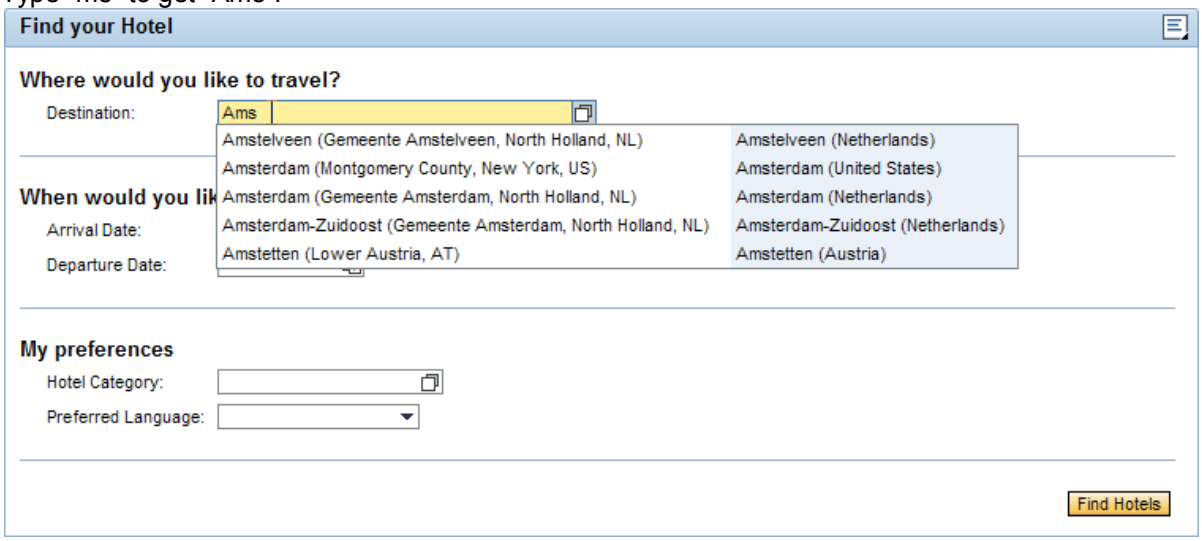
When you start the tutorial application, you should see the following screen:



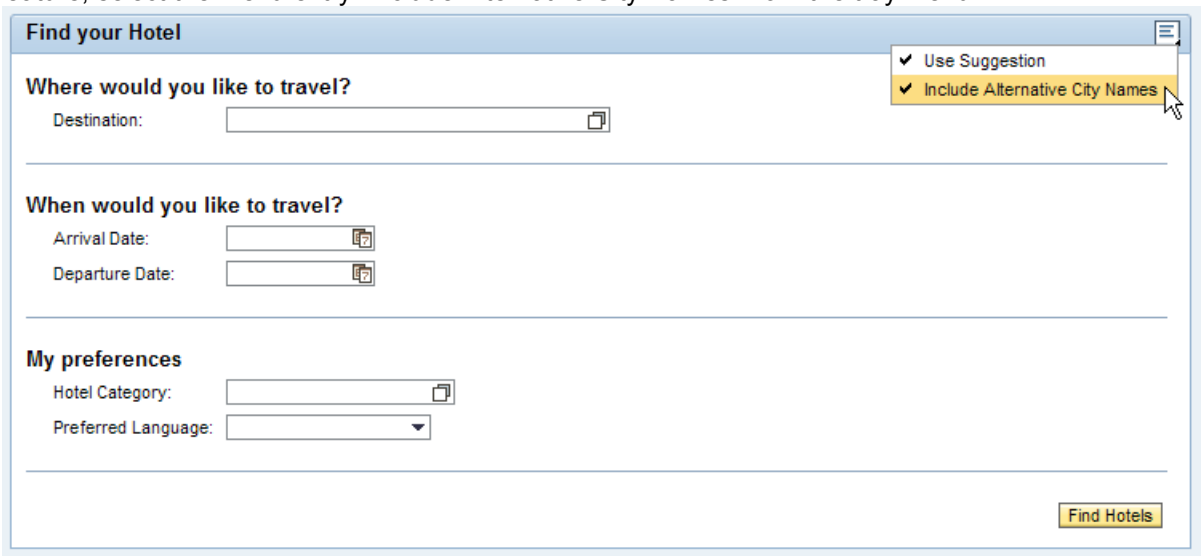
Type an uppercase "A" into the "Destination" field. A list of cities starting with "A" will be displayed below the field:



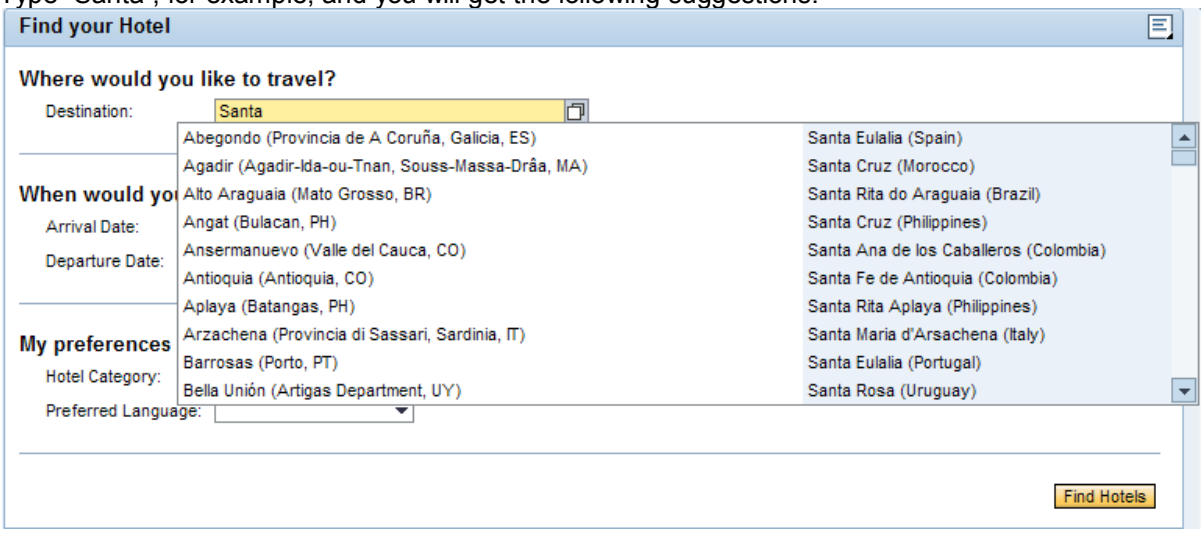
Type “ms” to get “Ams”:



You can extend the search for a city so that alternative city names are also included. To activate this feature, select the menu entry “Include Alternative City Names” from the tray menu:



Type “Santa”, for example, and you will get the following suggestions:



Note that the main city name of all these entries doesn't start with "Santa", but that there is an alternative name starting with that prefix.

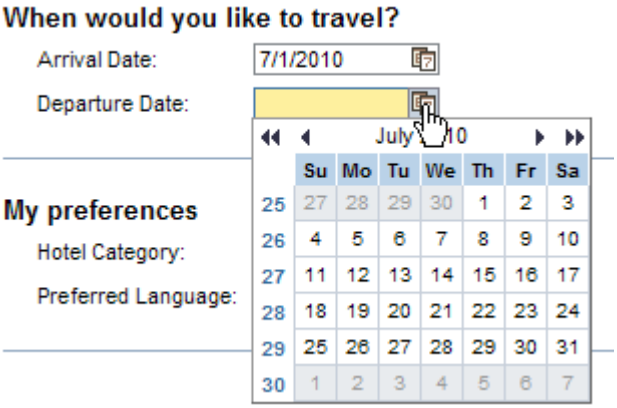
Open the OVS for the field and type "Santa" again; then press the "Go" button:

City	Alternative Name	Country	Region	District	Population
Abegondo	Santa Eulalia	Spain	Galicia	Provincia de A Coruña	5,711
Agadir	Santa Cruz	Morocco	Souss-Massa-Drâa	Agadir-Ida-ou-Tnan	698,310
Alto Araguaia	Santa Rita do Araguaia	Brazil	Mato Grosso		8,780
Angat	Santa Cruz	Philippines	Bulacan		41,235
Ansermanuevo	Santa Ana de los Caballeros	Colombia	Valle del Cauca		12,332
Antioquia	Santa Fe de Antioquia	Colombia	Antioquia		11,829
Aplaya	Santa Rita Aplaya	Philippines	Batangas		9,737
Arzachena	Santa Maria d'Arsachena	Italy	Sardinia	Provincia di Sassari	10,730
Barrosas	Santa Eulalia	Portugal	Porto		5,573
Bella Unión	Santa Rosa	Uruguay	Artigas Department		13,171

Note the additional column "Alternative Name" that contains the alternative city name matching the entered pattern.

Dependent Date Pickers

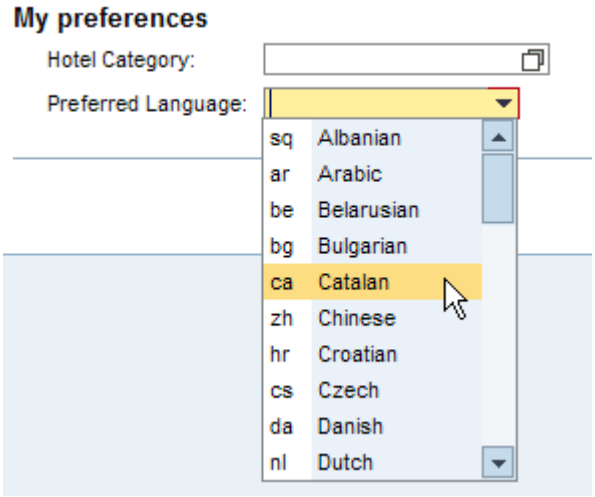
Select a date for the arrival, for example, July 1st, 2010. Then click on the calendar icon of the departure date field:



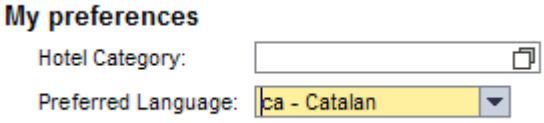
The date picker opens in the same year and month as the arrival date, as long as the departure field is still empty.

Value Selection with CctCode and Suggestion

Open the list of preferred languages by clicking the down-arrow:



Note that the input field is rendered like a dropdown list and that the list entries are rendered in two columns. Select an entry from the list.

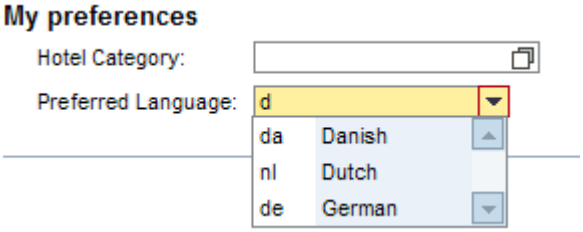


The input field displays the language key and the language name.

Enter a language name, for example, “English”, and press the ENTER key. The entered string is validated against the value set and, if the language exists, the key and language name are displayed inside the input field.

Enter some incorrect value such as “Anglish” and press ENTER. A validation error is displayed.

Clear the field again and type “d”. A list of suggested languages appears:



Note that the suggestion list contains also those entries where the language name starts with “d”, not only those where the key starts with “d”!

Restrictions

As always, the described features are only guaranteed to work for browser versions that are officially supported by the described Web Dynpro release.

More Information

SAP Developer Network SDN http://sdn.sap.com

Text Symbols

Symbol	Usage
	Note
	Recommendation
	Warning
	See also
→	Arrow for navigation paths

Copyright