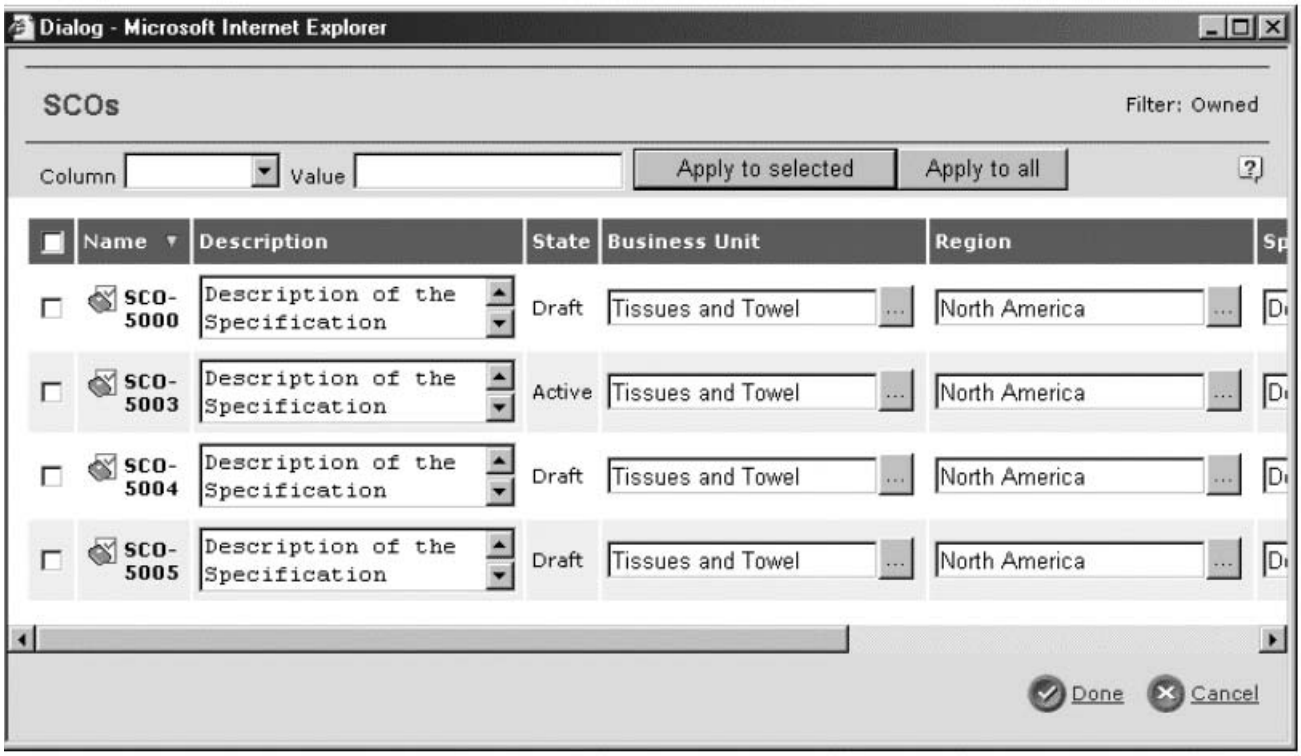
Editable Table

[The information in this prose should be added to the existing AEF section on building configurable tables.]

Configuring an Editable Table

Tables can be configured to display in an editable mode. For example, this graphic shows a standard view-only table that lists SCOs. When the user clicks the Edit link...

...the same table opens in a popup window but some column values are editable, as shown below.



Like form pages displayed in edit mode, you can configure which columns can be edited. For example, in the above graphic, the Name and State columns cannot be edited but the Description, Business Unit, and Region can.

After users change the data as needed, they save the data by clicking Done. If the column type is not businessobject or relationship, an update program and update function must be written and specified to update the table data.

The editable table does not support standard table controls such as the top and bottom action bar or toolbar, pagination controls, or filters. The currently-selected filter is displayed, as shown above, but users cannot change the selection.

Mass Updates

Several or all rows on a page can be updated with a single operation.

1. Select the rows to update using the check boxes at the beginning of each row. This step can be skipped if this change will be applied to all rows displayed.
2. Select the **Column** to update from the combo-box in the header. The **Value** field will automatically refresh to display one of the following:
   - a text box to enter a parameter
   - a combo-box to select a value from a range of values
   - a text box with a date chooser

   This will mirror the controls displayed for the individual fields shown in the form in the body frame.
3. Click **Apply to selected** to apply the changes to only the selected items

   Or

   Click **Apply to all** to apply to all records displayed.

Mechanisms that Define an Editable Table

The table and its columns are defined using the same table admin objects used to define view-only tables. Similar to form fields, when defining a table column that should be editable, you need to specify additional settings to tell the system how to handle edits to the column. For example, you need to specify the type of input control, any special validation for the data, and any update program for it.

The configurable JSP emxTableEdit.jsp creates editable table pages. This JSP is called automatically when the Edit link is clicked on a table page. The Edit link is included whenever editLink=true is passed to emxTable.jsp, as described below.

Adding an Editable Table to an Application

You add an editable version of a table page by adding the following parameter to the URL that calls the table:

editLink=true

If not passed, the parameter is assumed to be false. When editLink=true is included, the table page includes an Edit link in the toolbar or action bar. This link is automatically configured to call emxTableEdit.jsp. The system automatically passes the suiteKey and header parameters to the jsp, using the same values defined for the emxTable.jsp. This

means the title for the editable version of the table is the same as the title for the view-only version.

In addition to passing this parameter, follow these steps to add an editable table:

- Make sure all columns that should be editable include the Editable=true setting and other settings that define the type of input control, validation, and update programs needed to edit the data. For a list of parameters and settings, see *Parameters and Settings*. For examples of different types of editable columns, see *Defining Editable Table Columns*.
- Write any programs needed to get data or update data and create program objects to represent them.

Parameters and Settings

[Add the following to the section on URL Parameters for emxTable.jsp.]

Parameter	Description	Accepted Input Values
editLink	Use to display the Edit action link on the table page. The Edit link is automatically configured to call the editable version of the current table using emxTableEdit.jsp.	true false (default)

[Add the following rows to the table that lists settings for table columns, section "Parameters and Settings for Table Objects"]

Setting	Description	Accepted Input Values
Editable	Use to indicate whether the column is displayed as editable or read only. Only applies for Edit mode and View mode ignores the setting.	true (default)—Users can edit the column data when shown on the Edit mode. false—Users cannot edit the column data when shown in the Edit mode. The column looks just like it does in View mode except it is never hyperlinked.
Field Type	This setting is used for several purposes: • To indicate that the columns data should be obtained from a program or image instead of an expression. • When the data is obtained from an expression, if the data is basic information or an attribute. The system needs to know whether a column's data is basic information or an attribute in order to update the information correctly. Specifying whether the column is basic or an attribute is only required for columns that will be editable.	The Field Type can be one of the following: • program—The values for this column are obtained from a program (JPO). With this setting, the program and function name are required as settings. • programHTMLOutput—Same as the "program" setting above, except the column value output is in HTML format. Column values are placed in the table cell between <td> and </td> tags. This setting ignores other settings such as Show Type Icon, Href, format, and Alternate OID expression. • image—The value is an image. To specify the image file, use the Image setting. • basic—The column displays basic information for the business object. Basic information includes name, type, originated, policy, etc. Specifying basic as the field type is only needed when the column is editable. The only editable basic information is: type, name, revision, current, policy, description, owner, vault. • attribute—The column displays values for an attribute on the business object, such as Originator or Weight.

Setting	Description	Accepted Input Values
Input Type	Only used for tables in Edit mode. Specifies the type of HTML control to display for user input. You can also designate the size of the input boxes to allow for appropriate spacing. This is important for short numeric entry fields.	textbox—This is the default. The column has a single-line box for typing text. textarea—The column has a multi-line box for typing text. combobox—The column has a drop-down list of options and users can only select one value. Use combo boxes for attributes that have defined ranges and for attributes whose ranges are determined with a Range Helper URL. To see an example of a field with a combo box, see *Column Values Editable from Combo Box, Values from JPO*. popup—The column's data is filled in using a chooser or calendar in a popup window.
Range Function	Use to specify the name of the method in the JPO specified in the Range Program setting. See the Range Program setting below for more information.	The name of a function in the Range Program JPO, such as: • getAssignedRange • getClassificationRange • getPartUOM
Range OID Function	The method in the Range Program JPO that returns Id of the choices.	Name of method.
Range Program	Use to specify the name of a JPO that contains a method to get the column value ranges (choices). A range program is used only when the Input Type setting is combobox or popup. To see an example of a field configured with a range program, see *Column Values Editable from Combo Box, Values from JPO*. Also see *Writing JPO for Getting Values and OIDs for Combo Box*.	The name of a JPO, such as: • SCSBuyerDeskForm • TMCPackagesForm • ENCPartForm • AEFUtilForm

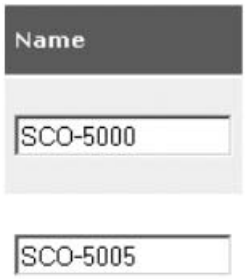
Setting	Description	Accepted Input Values
Update Function	Specifies a method name in the JPO given in the Update Program setting.	The name of a function in the Update Program JPO, such as: • setAssignedBuyerDesk • setPackage Access • setPartClassification
Update Program	Specifies a JPO that contains a method to set the column value when the column is displayed on an Edit mode table and when the user clicks Done. This program is used only when the Field Type setting is program or programHTMLOutput. *Using an update program is recommended only when the Field Type is not attribute or basic.* See *Using JPO to Update Table Values* for more information.	The name of a JPO, such as: • SCSBuyerDeskForm • TMCPackagesForm • ENCPartForm • AEFUtilForm

Defining Editable Table Columns

This section shows examples of how to configure table columns to be editable.

Column Values Editable in Text Box

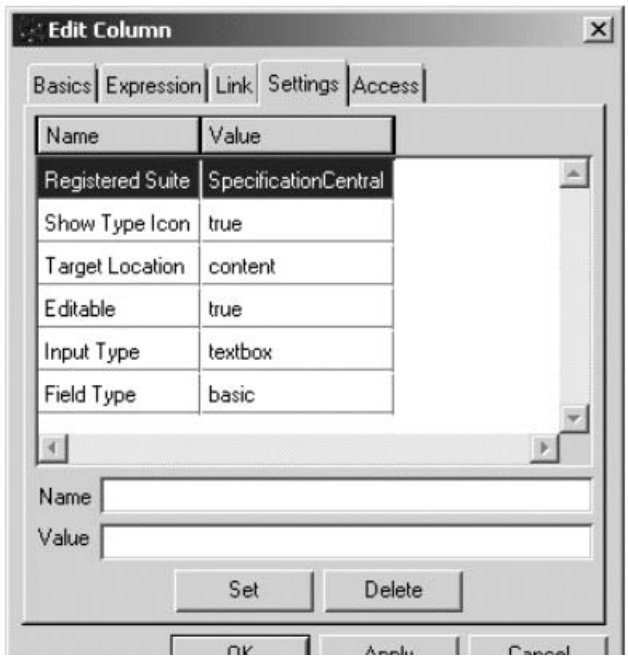
One way to let users edit column values is to let them type in new or changed information in a text box. For example, to get data for a Name column, you can use a select expression applied to the current business object and let users edit the name using a simple text box. This graphic shows examples of a Name column for a table displayed in Edit mode.



To configure a column so the data is obtained from a select expression and displayed in an editable text box, use these parameters and settings

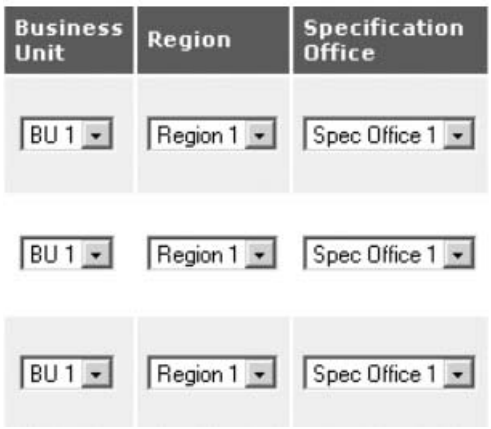
- Expression parameter—Enter the select expression that should be applied to a business object or relationship to get the column values.
- Applies to—Choose what the expression should be applied to: Business Object or Relationship.
- Field Type setting—When the expression is a business object basic or an attribute and the column is editable, you must include the Field Type setting. This lets the system correctly update the basic or attribute property.
- Editable setting—This setting must be true to let users edit the column data.

The below graphics show how the Name column shown previously is configured in Business Modeler.



Column Values Editable from Combo Box, Values from JPO

There are cases where a column can be edited but users shouldn't be able to just type in values. The values can only be changed to a specific set of values. If the set of possible values for the column is fairly small, they can be presented in a combo box and users can choose the value to change to. For example, the 3 columns shown below let users choose different values from combo boxes.

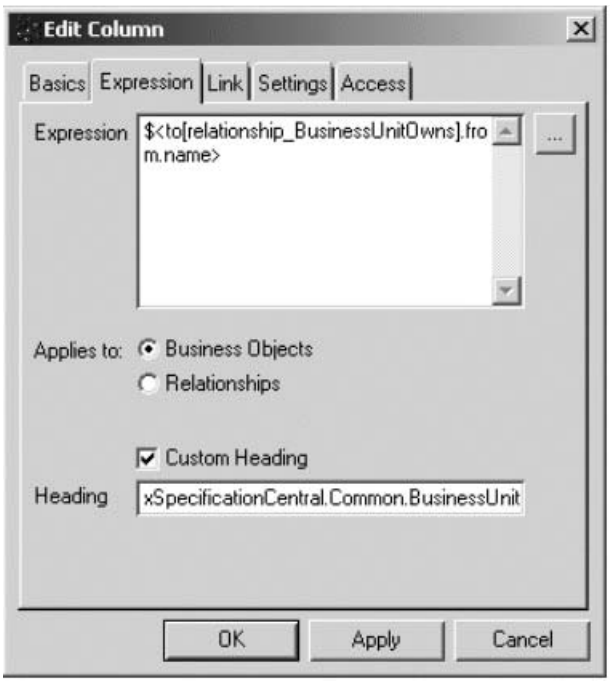


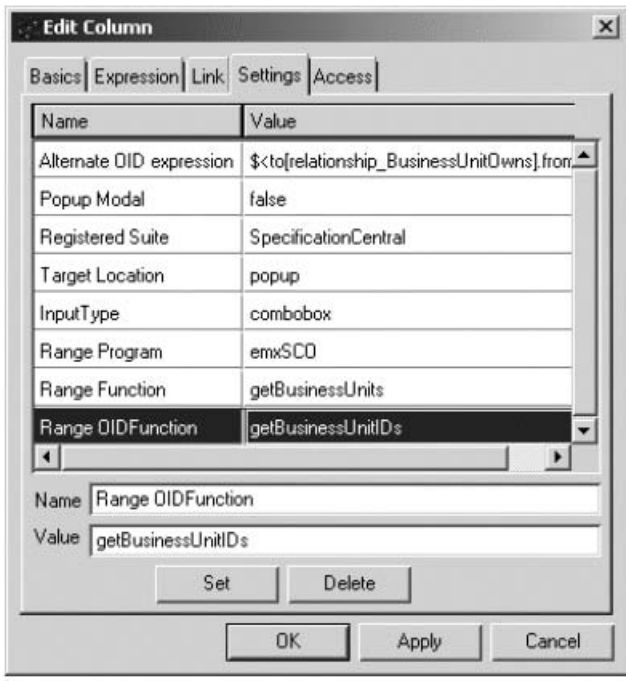
One way to get the values to populate the combo boxes with is to from a method in a JPO program. The method is defined to obtain the values based on specific business logic.

To display column values using a method in a JPO, use these parameters and settings

- Field Type setting—Set to `program` when program results contain only column values. Set to `programHTMLOutput` when program results contain complete HTML tag (including the column value) to be displayed in the column.
- Input Type setting—combobox
- Range Program setting—The JPO from which the choices are obtained.
- Range Function—The method in the Range Program JPO that returns the choices in the object type string list.
- Range OID Function—The method in the Range Program JPO that returns Id of the choices.
- Editable setting—This setting must be true to let users edit the column data.

These graphics show a column configured to display editable choices in a combo box, where the choices are retrieved from a JPO. The currently-selected value for the field, which is displayed on the view-only version of the table, is retrieved from an expression.





Column Values as Editable Dates

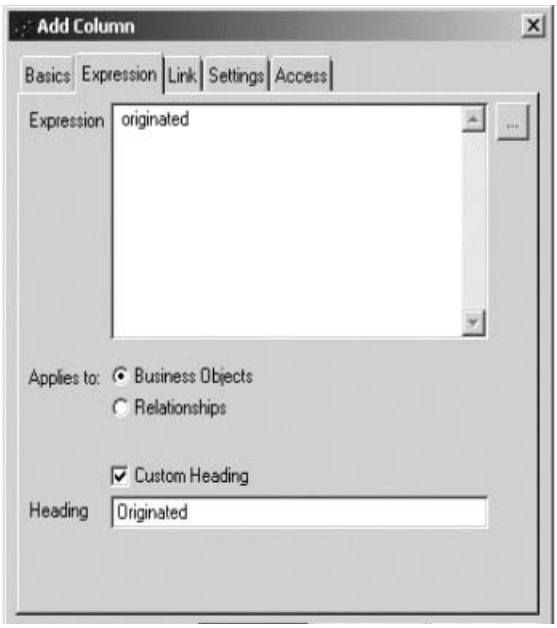
Column values can be configured to display dates and to let users change the date using a calender chooser. The value must be a valid date string. The date value is displayed based on the system and browser setting using the “lzDate” taglib. For example:

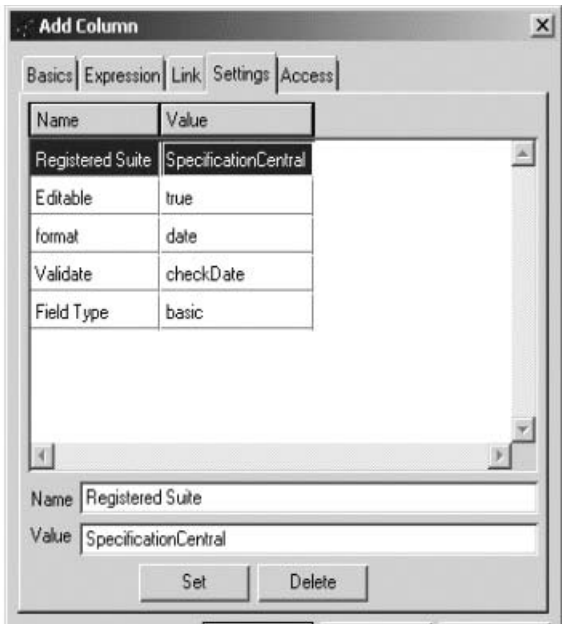
The format of the value displayed in the column should match the format listed in parenthesis following the date control.

To define a column as an editable date column, use these settings

- format setting—date
- Editable setting—true

For example, the following graphics show a column set up to display the originated date for the current object. The column is also set to validate the data.



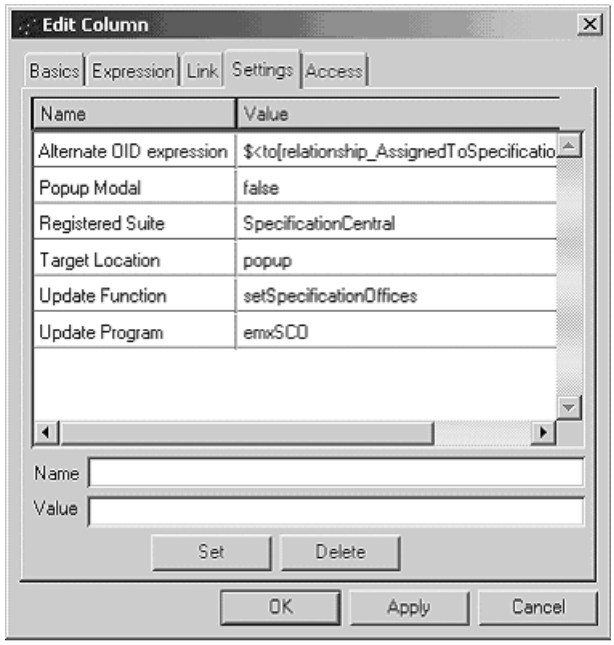


Using JPO to Update Table Values

To store the values of the related objects, JPOs can be used for saving the values. See *Writing JPO for Updating Field Values*.

The following are the two column settings that need to be saved through JPOs:

- Update Program—The JPO that contains the method that saves the values.
- Update Function—The name of method in the JPO that saves the data.



Writing JPO for Getting Values and OIDs for Combo Box

This JPO approach is used for obtaining the field values, when the form field is configured with the settings:

- Field Type=program
- program=JPO name
- function=JPO method name

For processing the data, the input parameter may be used by the method. The return type of this method is a StringList, which will contain one or more field value to be displayed.

The JPO method template is provided below:

```
public static Object methodName(Context context, String[] args) throws Exception
{
    HashMap programMap = (HashMap) JPO.unpackArgs(args);

    // Get the required parameter values from  "programMap" - as required
    String  objectId = (String) programMap.get("objectId ");
    String  relId = (String) programMap.get("relId ");
    HashMap requestMap = (String) programMap.get("requestMap");
    String  languageStr = (String) programMap.get("languageStr");

    // initialize the return variable
    StringList fieldValues = new StringList();

    // define and add selects if required

    // Process the information to obtain the values
    if ( objectId != null)
    {
```

```
    //add the field values
    fieldValues.addElement(…);

    }

    return fieldValues;
}
```

Sample JPO for Getting Field Values

The following is a sample Java code (JPO program object) for getting the current vault for the current object in the form. The method `getVault` in this program processes the object and returns the vault value.

Note: The vault name can be obtained by configuring the form field as businessobject select expression “vault” instead of defining the field Type as “program”. The “vault” example is used for to illustrate the steps involved in writing the JPO for a form field.

```
/*
 * emxUIFormSample
 *
 * Copyright (c) 1992-2003 MatrixOne, Inc.
 *
 * All Rights Reserved.
 * This program contains proprietary and trade secret information of
 * MatrixOne, Inc.  Copyright notice is precautionary only and does
 * not evidence any actual or intended publication of such program.
 *
 * static const char RCSID[] = $Id: Exp $
 */

import matrix.db.*;
import matrix.util.*;
import java.io.*;
import java.util.*;
import com.matrixone.framework.beans.*;
import com.matrixone.framework.util.*;
import com.matrixone.framework.ui.*;

/**
 * @version AEF 9.5.0.0 - Copyright (c) 2002, MatrixOne, Inc.
 */
public class ${CLASSNAME}
{

    /**
     *
     * @param context the eMatrix <code>Context</code> object
     * @param args holds no arguments
     * @throws Exception if the operation fails
     * @since AEF 9.5.0.0
```

```
 * @grade 0
 */
public ${CLASSNAME} (Context context, String[] args)
    throws Exception
{
    if (!context.isConnected())
        throw new Exception("not supported on desktop client");
}

/**
 * This method is executed if a specific method is not specified.
 *
 * @param context the eMatrix <code>Context</code> object
 * @param args holds no arguments
 * @returns nothing
 * @throws Exception if the operation fails
 * @since AEF 9.5.0.0
 */
public int mxMain(Context context, String[] args)
    throws Exception
{
    if (!context.isConnected())
        throw new Exception("not supported on desktop client");
    return 0;
}


/**
 * get Vault for the object.
 *
 * @param context the eMatrix <code>Context</code> object
 * @param args holds the following input arguments:
 *        0 - HashMap programMap
 * @returns StringList containing Vault name
 * @throws Exception if the operation fails
 * @since AEF 9.5.0.0
 */
public static Object getVault(Context context, String[] args)
    throws Exception
{
  HashMap programMap = (HashMap) JPO.unpackArgs(args);

String objectId = (String)programMap.get("objectId");
   StringList vaultList = new StringList();
  StringList listSelect = new StringList(1);
   listSelect.addElement("vault");

   if ( objectId != null)
   {
```

```
            BusinessObject bo = new BusinessObject(objectId);
            bo.open(context);
            String vault = bo.getVault();
            bo.close(context);
            vaultList.addElement(vault);
        }

        return vaultList;
    }


   /**
     * set Vault for the objects.
     *
     * @param context the eMatrix <code>Context</code> object
     * @param args holds the following input arguments:
     *        0 - HashMap programMap
     * @returns vector of Vault names
     * @throws Exception if the operation fails
     * @since AEF 9.5.0.0
     */
    public static int setVault(Context context, String[] args)
        throws Exception
    {
      HashMap programMap = (HashMap) JPO.unpackArgs(args);

    String objectId = (String)programMap.get("objectId");
    String newValue = (String)programMap.get("New Value");

        if ( objectId != null && newValue != null)
        {
            // Vault newVault = new Vault(newValue);
            BusinessObject busObj = new BusinessObject(objectId);
            busObj.open(context);
            // busObj.setVault(context, newVault);
            BusinessObject newBusObj = busObj.change(context, busObj.getTypeName(),
busObj.getName(), busObj.getRevision(), newValue, busObj.getPolicy().toString());
            busObj.close(context);
        }

        return (0);
    }




    /**
     * get getOriginatorRange for the field.
     *
     * @param context the eMatrix <code>Context</code> object
```

```
     * @param args holds the following input arguments:
     *        0 - HashMap programMap
     * @returns StringList of Range values
     * @throws Exception if the operation fails
     * @since AEF 9.5.0.0
     */

    public static Object getOriginatorRange(Context context, String[] args)
        throws Exception
    {
      HashMap programMap = (HashMap) JPO.unpackArgs(args);

    String objectId = (String)programMap.get("objectId");
    // HashMap paramMap = (HashMap)programMap.get("paramList");
        System.out.println("objectId << " + objectId + " >>");

       StringList rangeList = new StringList();
       rangeList.addElement("Test SeniorDesignEngineer");
       rangeList.addElement("Test Everything");
       rangeList.addElement("Test DesignEngineer");
       rangeList.addElement("Test Buyer");


        System.out.println("rangeList << " + rangeList + " >>");

       return rangeList;
    }

}
```

Writing JPO for Updating Field Values

This JPO approach is used for obtaining the field value ranges, when the form field is configured with the settings:

- Input Type=combobox / radiobutton / checkbox
- Range Program=JPO name
- Range Function=JPO method name

For processing the data, the input parameter may be used by the method. The return type of this method is a Boolean, which will be true if the update is successful. Otherwise it will be false.

The JPO function template is provided below:

```
public static int methodName(Context context, String[] args) throws Exception
{
      HashMap programMap = (HashMap) JPO.unpackArgs(args);

// Get the required parameter values from  "programMap" - as required
String  objectId = (String) programMap.get("objectId ");
   String  relId = (String) programMap.get("relId ");
```

```
HashMap requestMap = (String) programMap.get("requestMap");
String  languageStr = (String) programMap.get("languageStr");
String  newValue = (String) programMap.get("New Value");
String  oldValue = (String) programMap.get("Old Value");

// Define and add selects if required

   // Process the information to set the field values for the current object
….

   return 0;
}
```

See *Sample JPO for Getting Field Values.*