Configurator Intelligent Product Numbers based on Customer Selections

Brian J Looman
Greybrooke Consulting, Inc.

Introduction

Oracle Configurator is a very flexible and dynamic tool for building complex customer-specific products. Using this same flexibility, along with an advanced Configurator Developer extension, these customer selections can be used to generate and reuse intelligent product numbers. This greatly improves a company's ability to manage and maintain even the most complex inventory systems.

Challenge

Oracle Configurator can be a great tool for creating customer-specific items. However, the amount of items created by Configurator can get out of hand, and the item numbers will seldom be meaningful.

Solution

Using Oracle Configurator Developer, businesses can plug in a Configurator extension and setup text features that are associated to different components and options to implement intelligent item numbering. When a customer orders these items, the selections they choose will dynamically build the item number that will be used when fulfilling the order.

Benefits

- Build product numbers based on the components selected by a customer.
- Manage the item master by validating against existing items and reusing items when available.
- Implement intelligent item numbers that are both meaningful and reusable.

Implementation

In order for Oracle Configurator to build this product number, first a set of text features will need to be created to implement the intelligent item number logic. In general, each of the text features should be prefixed with an identifier that the Configurator extension can use to distinguish the related text features.

A Text Feature must be created to store the generated item number at the model level for the given product category/type. This Feature should be a default item number that can be built upon. As a customer chooses different components and options, this is the text feature that would be updated will the appropriate item segment values.

Text Features must also be created and associated to components and options within the model. The default values of the text features follow a format similar to the following: **5:BA, 10:X77**. The text features contain a set of position and values for the different segments that will make up the intelligent product number. Therefore, when a customer selects this specific option, the added Configurator extension will replace the item segment values at the specific positions.

Here is a snippet of the Configurator extension used to implement this logic:

```
private void buildItemNumber(oracle.apps.cz.cio.Configuration configuration)
        throws Exception {
  Component rootComponent;
  TextNode itemNumTextNode;
  String itemNumber;

  // Get the configuration root node.
  rootComponent = configuration.getRootComponent();

  // Get the Item Number Text Feature.
  itemNumTextNode = rootComponent.getChildByName ("XXX-ITEM NUMBER");

  // Loop through the Item Nodes starting at the root.
  //   When any of the item number text features are found,
  //   this method will update the default item number
  //   value with new segment values.
  itemNumber = loopItemNodes(rootComponent,itemNumTextNode);

  // Set the Text Feature value to the new item number.
  itemNumTextNode.setTextValue(itemNumber);
}
```

This extension will be executed as a Rule on the preConfigSave event in Configurator Developer.

After the user has booked the order, there is a database package that retrieves the product text feature from the configuration. It uses a standard package that is provided by Oracle for producing custom item numbers. This package also manages the creation of items, and will reuse an item if the item number is the same.

```
CREATE OR REPLACE package body APPS.BOMPCFGI as

  FUNCTION user_item_number ( model_line_id  IN  NUMBER )
   RETURN VARCHAR2 IS
    l_item_number    VARCHAR2(40)   DEFAULT NULL;

    CURSOR c_item (cp_line_id  IN  NUMBER) IS
      SELECT cci.item_val
        FROM apps.oe_order_lines_all ool,
             apps.cz_config_items cci
       WHERE ool.config_hdr_id = cci.config_hdr_id
         AND ool.config_rev_nbr = cci.config_rev_nbr
         AND ool.line_id = cp_line_id
         AND cci.ps_node_name  = 'XXX-ITEM NUMBER';
  BEGIN
    -- ------------------------------------------------------
    -- Fetch the text feature containing the item number
    --   for this order line's configuration.
    -- ------------------------------------------------------
```



```
    OPEN c_item ( cp_line_id => model_line_id );
    FETCH c_item INTO l_item_number;
    CLOSE c_item;

    RETURN l_item_number;
  END user_item_number;

END BOMPCFGI;
```

Once all these pieces are in place, you can use Configurator Developer to test the user interface. Once the item number logic is validated, publishing the configurations will allow both Order Management and iStore to utilize the functionality.

About the Author

Brian Looman is Director of Technologies for Greybrooke Consulting, a firm specializing in Oracle Applications implementations and upgrades with principal offices in Orlando, Atlanta, Chicago, and Tampa. Brian has a unique combination of business and technical expertise that is seldom seen in the ERP consulting arena. He has both a keen sense for business and a wealth of experience as a solutions architect, functional analyst, and technical lead.

His focus is on improving client ERP systems using a combination of technical and functional expertise in both product implementations and solution designs. He has demonstrated his functional expertise for over eleven years across a variety of Oracle Applications and business processes - Process & Discrete Manufacturing, Order-to-Cash, and Procure-to-Pay. In addition, his technical expertise spans practically all Oracle development tools and technologies.

More papers and presentations written by Brian are available at www.greybrooke.com.