

Oracle 11i – Extending Processing Constraints beyond what oracle promises.

A trick to give more friendly user messages using Processing Constraints, and to use processing constraints for giving warning.

```
/*
```

This example explains

- a) how to use Standard Oracle Messages to give use a variable in a processing constraint (hence giving more meaningful error).
- b) How to give Warning (but not allow the transaction to proceed) using processing constraints)

All the comments are highlighted in Grey.

The crucial parts of the code are highlighted in yellow.

This example illustrates a custom Processing Constraint for Minimum Order Quantity.

It assumes that The Minimum Order Quantity for any Item/Warehouse is stored in attribute1 of mtl_system_items.

The requirement is to stop (or just give a warning) if the ordered quantity is less the Minimum Order Quantity for the given item and warehouse and to give the user an appropriate message. If we were to use the standard Processing Constraint for this. The error would be something like “Ordered Quantity is less than Minimum Order Quantity”, but there would be now way to tell the user what the minimum order quantity is. This example shows how we can give that information to the user as well.

This example also shows how you could just give a popup message warning to the user, and not stop the transaction.

```
*/
```

```
CREATE OR REPLACE PACKAGE Xxxx_Processing_Cons_Api
AS
```

```
  PROCEDURE enforce_order_quantities
    (p_application_id      IN NUMBER,
     p_entity_short_name   IN VARCHAR2,
     p_validation_entity_short_name IN VARCHAR2,
     p_validation_tmplt_short_name IN VARCHAR2,
     p_record_set_tmplt_short_name IN VARCHAR2,
     p_scope IN VARCHAR2,
     p_result OUT NUMBER );
```

```
END Xxxx_Processing_Cons_Api;
```

```
/
```

```
show errors package Xxxx_Processing_Cons_Api;
```

```
CREATE OR REPLACE PACKAGE BODY Xxxx_Processing_Cons_Api
AS
```

```
  PROCEDURE enforce_order_quantities
    (p_application_id      IN NUMBER,
     p_entity_short_name   IN VARCHAR2,
     p_validation_entity_short_name IN VARCHAR2,
     p_validation_tmplt_short_name IN VARCHAR2,
     p_record_set_tmplt_short_name IN VARCHAR2,
```

```

        p_scope          IN VARCHAR2,
        p_result         OUT NUMBER )
IS
    x_organization_id   NUMBER;
    x_master_organization_id NUMBER;
    x_item_validation_org NUMBER;
    x_minimum_order_quantity NUMBER;
    x_increments        NUMBER;
    x_split_exists      VARCHAR2(1);
    x_pass              EXCEPTION;
    err_msg             VARCHAR2(2000);

BEGIN

    -- Default.
    p_result := 0;

    Oe_Debug_Pub.ADD('-----Entering Enforce_Order_Quantity -----', 3);
    Oe_Debug_Pub.ADD('validation entity name:'||p_validation_entity_short_name);

    SELECT master_organization_id
    INTO   x_master_organization_id
    FROM   oe_system_parameters;

    /*
    Oe_Line_Security.g_record is the structure to be used to get all the order/line data in the memory
    */

    x_organization_id := NVL(Oe_Line_Security.g_record.ship_from_org_id, x_item_validation_org);

    Oe_Debug_Pub.ADD('Inventory Item : '|| Oe_Line_Security.g_record.inventory_item_id);
    Oe_Debug_Pub.ADD('Master Organization Id : '|| x_master_organization_id);
    Oe_Debug_Pub.ADD('Ordered Quantity ' || Oe_Line_Security.g_record.ordered_quantity);

    BEGIN

        IF NVL(Oe_Line_Security.g_record.ordered_quantity, 0) <= 0 THEN
            RAISE x_pass;
        END IF;

        IF NVL(Oe_Line_Security.g_record.line_category_code, 'X') = 'RETURN' THEN
            RAISE x_pass;
        END IF;

    /*
    Assumption here is that this function will return Minimum Order Quantity from where ever it is stored
    */

```

```

*/
select    attribute1
into      x_minimum_order_quantity
from      mtl_system_items
where     inventory_item_id = Oe_Line_Security.g_record.inventory_item_id
and       organization_id = x_organization_id;

```

```
Oe_Debug_Pub.ADD('MIN ORDER Qty : ' || x_minimum_order_quantity , 3);
```

```

/*
This section merely ensures that Minimum Order Quantity rule is not applied to split lines or to returns.
*/

```

```
-- Check if the line is part of a split.
```

```

IF Oe_Line_Security.g_record.line_set_id IS NOT NULL THEN
  BEGIN
    SELECT 'Y'
    INTO   x_split_exists
    FROM   oe_sets
    WHERE  set_id = Oe_Line_Security.g_record.line_set_id
    AND    set_type = 'SPLIT';
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      x_split_exists := 'N';
  END;

ELSE
  x_split_exists := 'N';
END IF;

IF x_minimum_order_quantity IS NULL
THEN
  RAISE x_pass;
ELSE
  IF (Oe_Line_Security.g_record.ordered_quantity < x_minimum_order_quantity AND
      x_split_exists = 'N')

  THEN

```

```

/*

```

This is really the crux of solution. Use FND MESSAGE to generate a custom message.

If you want the condition to be a hard constraint return 1 (for failure), if you want it to be merely a warning return 0 (pass, however the message would have already been displayed to the user).

You can create the error Message on the fly or have seed one. The message (Say) “Ordered Quantity is less than Minimum Order Quantity. Minimum Order Quantity is 5.” is so much more useful and meaningful to the user than just saying “Ordered Quantity is less than Minimum Order Quantity” and asking the user to figure out what the Minimum Ordered Quantity is.

```
*/
```

```
/*
```

```
-- to create message on the fly.
```

```
    Oe_Msg_Pub.Add_Text('The Ordered Quantity should be >= ' || x_minimum_order_quantity || ', AND IN increments OF '
|| x_increments || ' thereafter.');
```

```
*/
```

```

Fnd_Message.set_name('ONT', 'XXXX_OM_QUANTITY');
Fnd_Message.set_token('MOQ', x_minimum_order_quantity);
Fnd_Message.set_token('SPI', x_increments);
                                err_msg := Fnd_Message.get;
                                Oe_Msg_Pub.ADD_text(err_msg);
                                Oe_Debug_Pub.ADD(err_msg);

```

```
/*
```

```
For processing constraint.
```

```
*/
```

```
p_result := 1; -- Fail
```

```
/*
```

```
or If you just want to give a warning.
```

```
*/
```

```
raise x_pass;
```

```
END IF;
END IF;
```

```
Oe_Debug_Pub.ADD('Result IS : ' || p_result, 3);
Oe_Debug_Pub.ADD('-----Exiting Enforce_Order_Quantity -----', 3);
```

```
EXCEPTION
  WHEN x_pass THEN
    p_result := 0;
    Oe_Debug_Pub.ADD('Result IS : ' || p_result, 3);
    Oe_Debug_Pub.ADD('-----Exiting Enforce_Order_Quantity -----', 3);

  WHEN OTHERS THEN

    p_result := 0;
    Oe_Debug_Pub.ADD('Result IS : ' || p_result, 3);
    Oe_Debug_Pub.ADD('-----Exiting Enforce_Order_Quantity -----', 3);

END enforce_order_quantities;

END Xxx Processing_Cons_Api;
/
show errors package body Xxx Processing_Cons_Api;
```

About the Author

© Neocortex, Inc.

This document contains information that is proprietary to Neocortex, Inc. Do not reproduce or reuse this information without express written consent of Neocortex, Inc.

Pankaj Jain, is a founding partner of Neocortex. He has worked as an Oracle Applications consultant for more than 9 years now. He is an IIM, Ahmedabad alumnus. Please send any comments/remarks on this paper to him at pankaj@neocortex.com. We will love to find out if this document was useful to you, please let us know. If you want to implement this solution at your organization, please let us know, we can send you all the code we developed for prototyping this solution.



Neocortex, Inc.
39275, State Street,
Fremont, CA 94538
www.neocortex.com
info@neocortex.com
510-435-8536