Extending Advanced Pricing to Custom Applications

Chris Herdic
OappsNet

Abstract

This presentation will discuss techniques of extending standard Advanced Pricing functionality to integrate with custom applications. An overview of the Advanced Pricing module, using the PRICE_REQUEST API, mapping attributes to custom data sources, and use of GET_CUSTOM_PRICE will be discussed.

Objective

Oracle Application's Advanced Pricing (QP) module is a flexible, extendable module that provides a common source for setting up rules to derive prices. This paper will define key components of the Advanced Pricing module, present methods and identify components to extend aspects of the module to custom data, and discuss the risks and rewards of customization.

The objective of this paper is to present:

Concepts

- Discuss key components of Advanced Pricing
- Provide an overview of Advanced Pricing Architecture

Code

- Present Extendible Components of Advanced Pricing
    - Attribute Linking
    - Get_Custom_Price
    - Request_Price (Pricing Engine)
- Identify other Open APIs

Customization Considerations

- Customization Risk and Reward
- Customization methods
- Customization Do's and Don'ts

Advanced Pricing Concepts

The Advanced Pricing module was introduced with Oracle Applications Release 11i. Pricing in releases prior to R11i Oracle Applications was comprised of price list and pricing agreements. As a result, many implementations prior to R11i developed external custom applications to capture complex pricing structures to meet market demands. The basic 10.x pricing concepts were preserved and enhanced in R11i, while many new pricing components were introduced that enabled businesses to develop pricing models, rules, multi request systems that bridged functionality gaps that required prior customizations. R11i introduced a pricing tool that would meet the needs of both simple and complex pricing requirements, and, enable architecture to interface with data and business logic sourced outside of core Oracle Application modules.

The following pricing entities are important to define before the discussion of extending Advanced Pricing to custom applications.

Price Lists

The price list remains as the starting point for a pricing transaction. All orders must have a price, and the price list is where this price originates. Basic information on a price list includes the price list name, item, list price, currency, and effective dates. Additionally, list prices can be established for an item category. List prices can be derived from a defined value, static formula (calculated prior to requesting a list price) or dynamic formula (calculated at the time of a price request).

Qualifiers

Qualifiers define the rules of eligibility determining a price. The attributes of a transaction are evaluated to enable, or qualify, for a pricing entity (price lists and modifiers). Price lists and modifiers can be qualified by a qualifier attribute. A qualifier attribute is a data element that is derived from a component of the transaction being priced. It may be as simple as the item or customer on a transaction, or, a value returned by program logic.

Modifiers

Modifiers are the pricing discounts, benefits, surcharges, and terms adjustments that can be applied to an order when a price is requested. The Modifier also defines:

- Currency in which pricing is calculated
- Method to calculate discount (percent, lump sum, new price, amount)
- Level (header / line) the modifier will apply
- Pricing phase (when the modifier is to be considered)
- Effective Dates
- Item or Item Category that is eligible
- Bucket (sub total) from which a discount is applied
- Eligibility of the Modifier (Qualifiers)
- Incompatibility

For price discounts or charges, the application method, value, and formula can be defined to calculate the value.

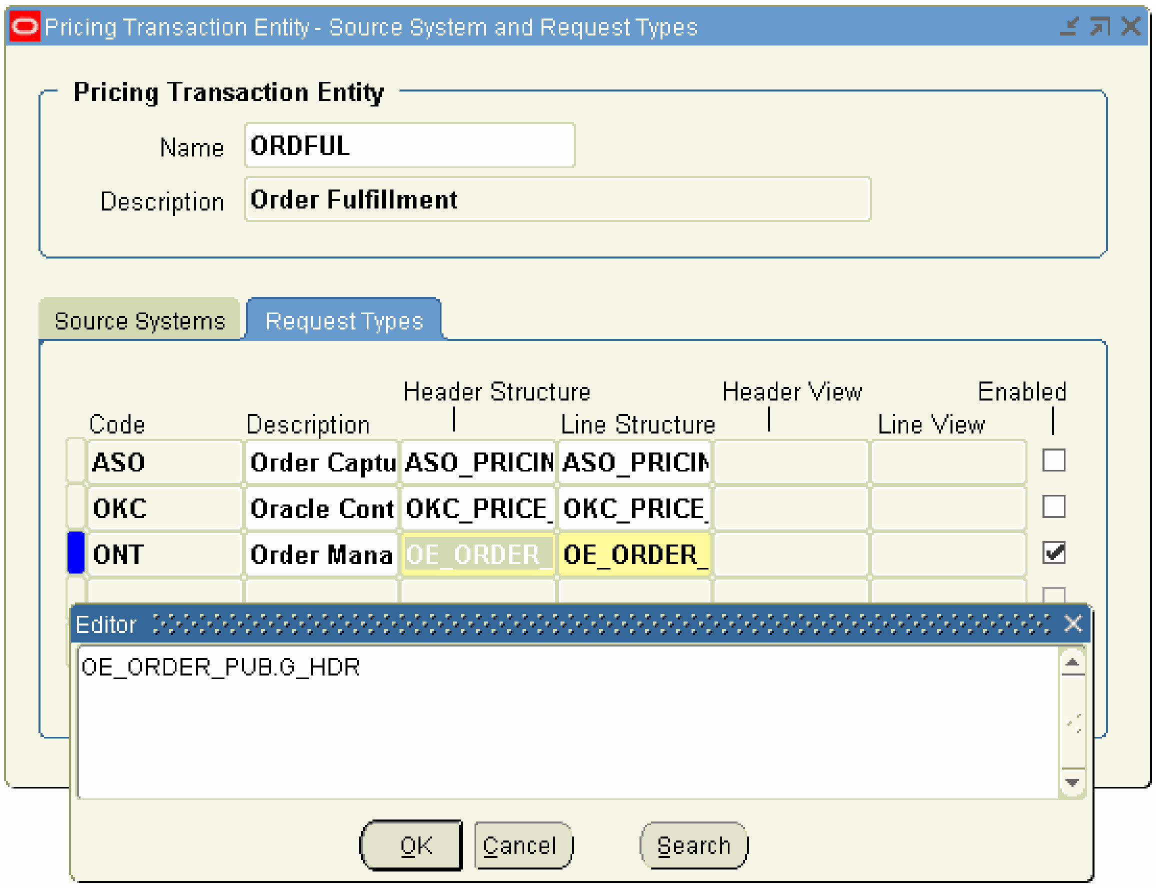
Formulas

A series of steps and mathematical expression interpreted and executed by the pricing engine to calculate list prices or modifier discounts. This is the Advanced Pricing extension point for GET_CUSTOM_PRICE as discussed in detail later in this paper.

Request Types

Request types define the header and line PL/SQL structures that are used to pass transaction information to the pricing engine. These structures are the entry point for price request processing. A group of request types for a similar transaction source is referred to as a Pricing Transaction Entity. Figure 1 displays how global structures are defined for a request type.

Figure 1



Pricing Transaction Entity - Source System and Request Types

Pricing Transaction Entity

Name: ORDFUL

Description: Order Fulfillment

Code	Description	Header Structure	Line Structure	Header View	Line View	Enabled
ASO	Order Captu	ASO_PRICIN	ASO_PRICIN			☐
OKC	Oracle Cont	OKC_PRICE_	OKC_PRICE_			☐
ONT	Order Mana	OE_ORDER_	OE_ORDER_			☑

Editor

```
OE_ORDER_PUB.G_HDR
```

Contexts and Attributes

The Attribute, in the Advanced Pricing Module, is a data element that captures values from the transaction used by the pricing engine to determine a price. This is the point that all transaction information is consolidated and passed to the pricing engine. The logical groupings of attributes are called contexts; for which there are three (3) types:

Qualifier Attributes

Qualifier attributes are used for the purpose of qualifying for a modifier or price list. Examples of seeded contexts (logical groupings) of qualifier contexts are 'CUSTOMER' and 'ORDER'. Qualifier attributes for each qualifier context are defined. The values, derived at the time of a price request, are held in the attribute for the pricing engine to use when determining a price. Custom qualifier contexts can be defined, and, custom attributes can be defined for seeded contexts or custom contexts. If a custom qualifier attribute is needed, begin using QUALIFIER_ATTRIBUTE31 or greater. Oracle reserves QUALIFIER_ATTRIBUTE1 to QUALIFIER_ATTRIBUTE30 for seeded context / attribute definitions.

Pricing Attributes

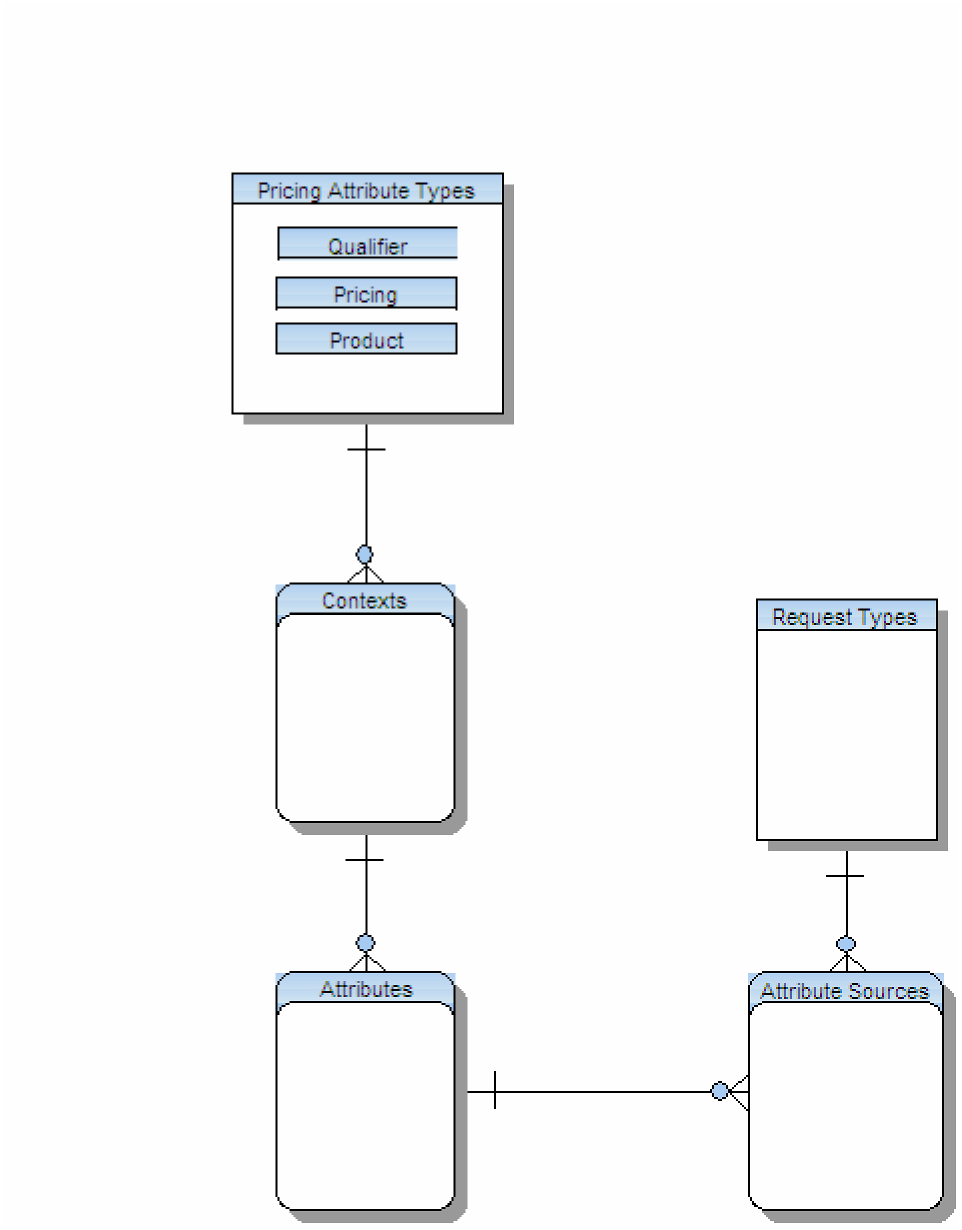
Pricing attributes are values that are captured the time a pricing transaction is entered, or, an additional way to define eligibility for a modifier or price list. The two distinguishing properties of a pricing attribute are that the values can be referenced in a formula line step, and, the values can be captured on the order line in the pricing context / attribute form item. Again, custom definition of pricing attributes should use PRICING_ATTRIBUTE31 or greater.

Product Attributes

Product attributes capture information related to an item. It is comprised of a single context: ITEM. Attributes defined for the ITEM contexts are derived from item categories and segments. Product attributes are used on modifier and price list lines to determine the type of item, item category, or hierarchy that is being priced. New custom contexts may not be created for product attributes; however, new attributes may be defined for an existing product context. Product attributes use the same physical columns as pricing attributes. PRICING_ATTRIBUTE31 and greater are reserved for custom definition.

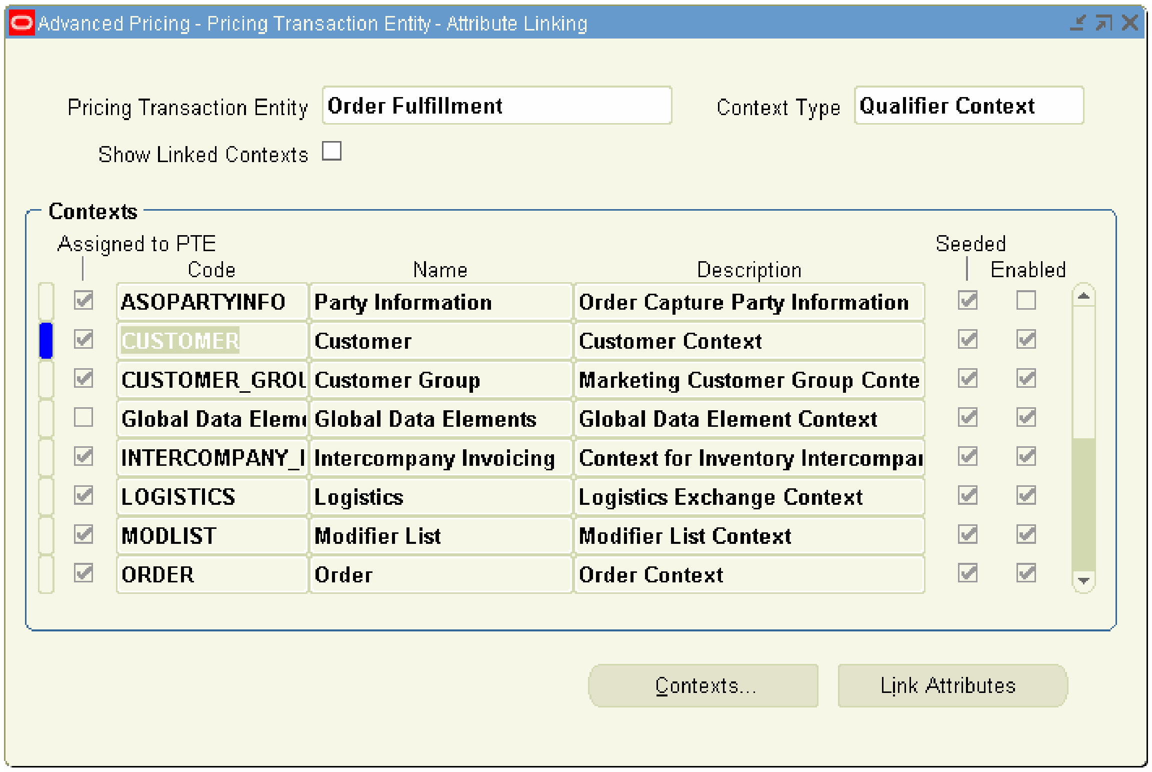
Figure 2 is a general entity relationship diagram describing the relationships of the Advanced Pricing module's context and attributes. It does not reflect how the data is physically stored.

Figure 2



Although technically, contexts and attributes are similar in structure to Application Object Library's descriptive flexfields, pricing contexts and attributes are defined and resolved at the time a request for price is executed in PL/SQL structures or global temporary tables.   When a request for price is initiated, the qualifier, product, and pricing attributes define the place that stores the values that the pricing engine uses to evaluate the pricing rules. The context logically groups the attributes; the attribute defines where a value is stored and the source of the value.   Figure 3 shows a list of qualifier contexts used for the order fulfillment Pricing Transaction Entity (PTE).

Figure 3



Advanced Pricing - Pricing Transaction Entity - Attribute Linking

Pricing Transaction Entity: Order Fulfillment

Context Type: Qualifier Context

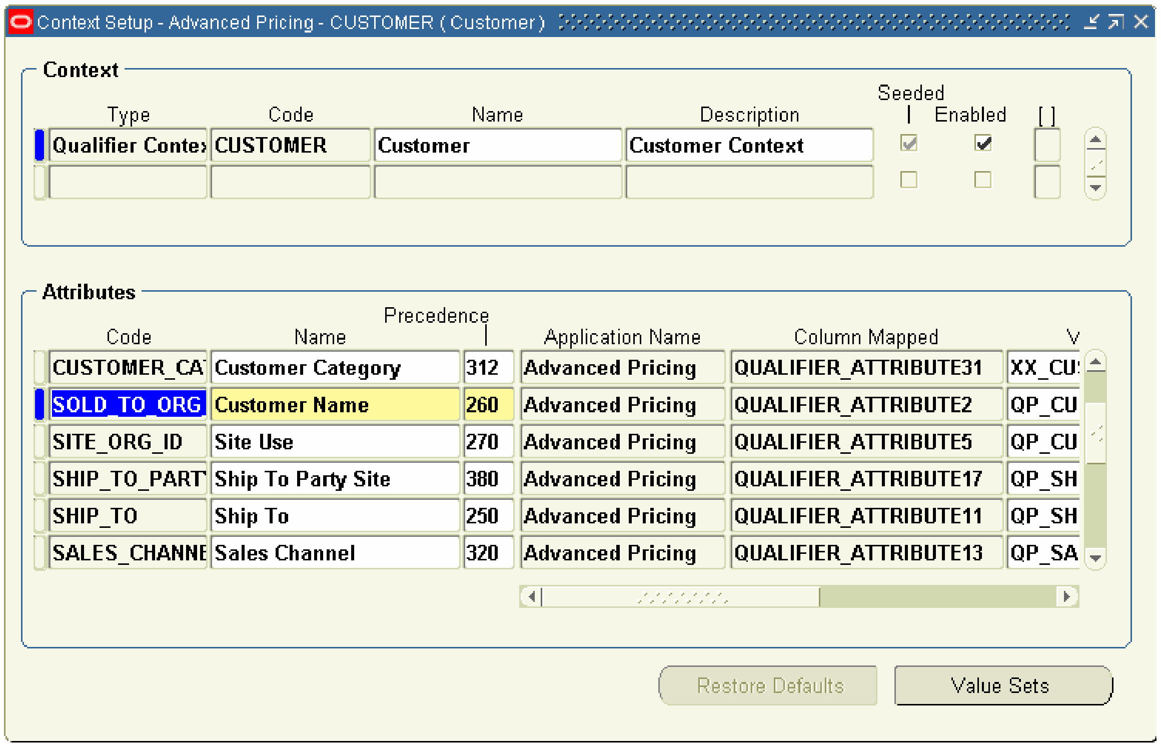
Show Linked Contexts: ☐

Contexts

Assigned to PTE	Code	Name	Description	Seeded	Enabled
✓	ASOPARTYINFO	Party Information	Order Capture Party Information	✓	☐
✓	CUSTOMER	Customer	Customer Context	✓	✓
✓	CUSTOMER_GROU	Customer Group	Marketing Customer Group Conte	✓	✓
☐	Global Data Elem	Global Data Elements	Global Data Element Context	✓	✓
✓	INTERCOMPANY_I	Intercompany Invoicing	Context for Inventory Intercompa	✓	✓
✓	LOGISTICS	Logistics	Logistics Exchange Context	✓	✓
✓	MODLIST	Modifier List	Modifier List Context	✓	✓
✓	ORDER	Order	Order Context	✓	✓

Figure 4 displays the CUSTOMER qualifier context, attribute definitions.

Figure 4


Context Setup - Advanced Pricing - CUSTOMER (Customer)

Context

Type	Code	Name	Description	Seeded	Enabled	[ ]
Qualifier Conte:	CUSTOMER	Customer	Customer Context	✓	✓	

Attributes

Code	Name	Precedence	Application Name	Column Mapped	V
CUSTOMER_CA	Customer Category	312	Advanced Pricing	QUALIFIER_ATTRIBUTE31	XX_CU:
SOLD_TO_ORG	Customer Name	260	Advanced Pricing	QUALIFIER_ATTRIBUTE2	QP_CU
SITE_ORG_ID	Site Use	270	Advanced Pricing	QUALIFIER_ATTRIBUTE5	QP_CU
SHIP_TO_PART	Ship To Party Site	380	Advanced Pricing	QUALIFIER_ATTRIBUTE17	QP_SH
SHIP_TO	Ship To	250	Advanced Pricing	QUALIFIER_ATTRIBUTE11	QP_SH
SALES_CHANNE	Sales Channel	320	Advanced Pricing	QUALIFIER_ATTRIBUTE13	QP_SA

Each qualifier attribute now must define the source for the value at the time a transaction is priced. When a request for price is executed, a VALUE for all CONTEXT / ATTRIBUTE pairs is resolved. These values are used by the pricing engine for that pricing transaction request. The CONTEXT / ATTRIBUTE / VALUE combinations are stored in PL/SQL or temporary table structures as a source for the pricing engine to determine a price. To define the source, the attribute must identify how the value is derived.

Attribute Linking

Linking an attribute defines the source of a VALUE for the CONTEXT / ATTRIBUTE combination. There are three methods to link an attribute to a source:

User Entered
The value that is captured in a pricing context attribute and is entered on the transaction. For Order Management, this is the pricing context field under the pricing tab. The value of this mapping is simply derived from the PRICING_ATTRIBUTE on OE_ORDER_LINES_ALL.

Custom Sourced
PL/SQL logic for custom sourced attribute is stored and compiled in an Oracle provided package procedure: *QP_CUSTOM_SOURCE. Get_Custom_Attribute_Values*. In this API, the user defined PL/SQL logic

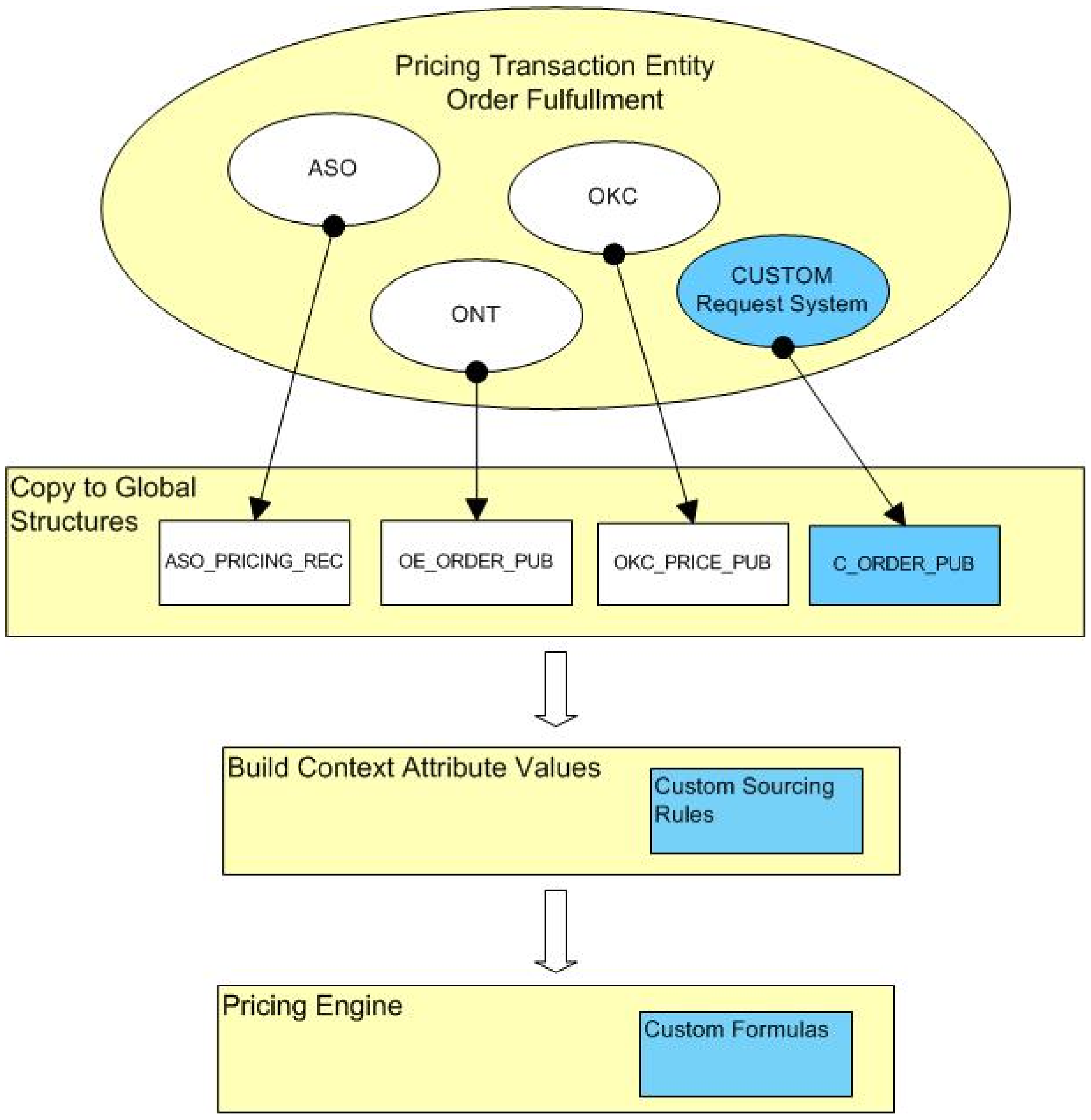
derives a value and explicitly assigns the CONTEXT, ATTRIBUTE, VALUE for a request type (ONT, ASO) and pricing type (**H**eader or **L**ine).

Attribute Mapping
The attribute value is determined for the header level, line level, or both levels of the priced transaction by a value derived from a profile option, PL/SQL API, or PL/SQL Multi-Record. In the Attribute Mapping method, the assignment of the CONTEXT, ATTRIBUTE, VALUE combination is made by the BUILD_SOURCING API. The PL/SQL function contains logic to derive a value.

Extendable Components of Advanced Pricing Architecture

The Advanced Pricing Architecture provides many opportunities to extend to custom applications and custom data sources. Figure 5 documents the extendable components of the Advanced Pricing architecture that extend during the request for a price.

Figure 5



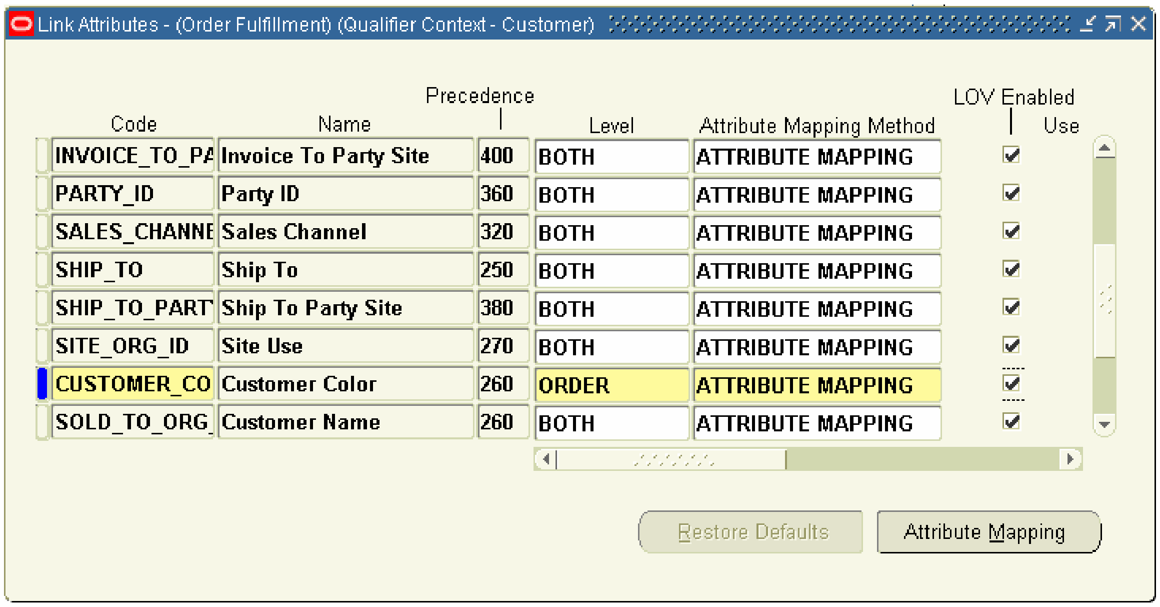
Custom Sourcing Rules - Attribute Linking

Linking an attribute is the most common way to extend Advanced Pricing to custom data sources. This feature provides the ability to develop logic that uses information captured on a pricing transaction derive a value that can be used to qualify for a pricing modifier or price list. The logic developed and executed by a request for price may also be used by other systems requiring the same value. The logic developed can derive values from data within Oracle Applications, or, from data maintained in custom applications.

To link an attribute to a custom PL/SQL package, navigate to the Attribute and Linking Form, select Attribute Mapping , the Request System that will access this mapping, and the level at which the mapped value will be applied (LINE, HEADER, BOTH). Select

PL/SQL Source and enter the PL/SQL *package_name.function_name* and parameters needed by the function to execute. The parameter values must reside on the PL/SQL global header or line structure. These values have been copied by the calling application to the global structures. Figure 6 displays a custom qualifier attribute: Customer Color, that a value will be sourced by an Attribute Mapping from an Order Fulfillment (ONT) transaction, with values derived from the order header.

Figure 6

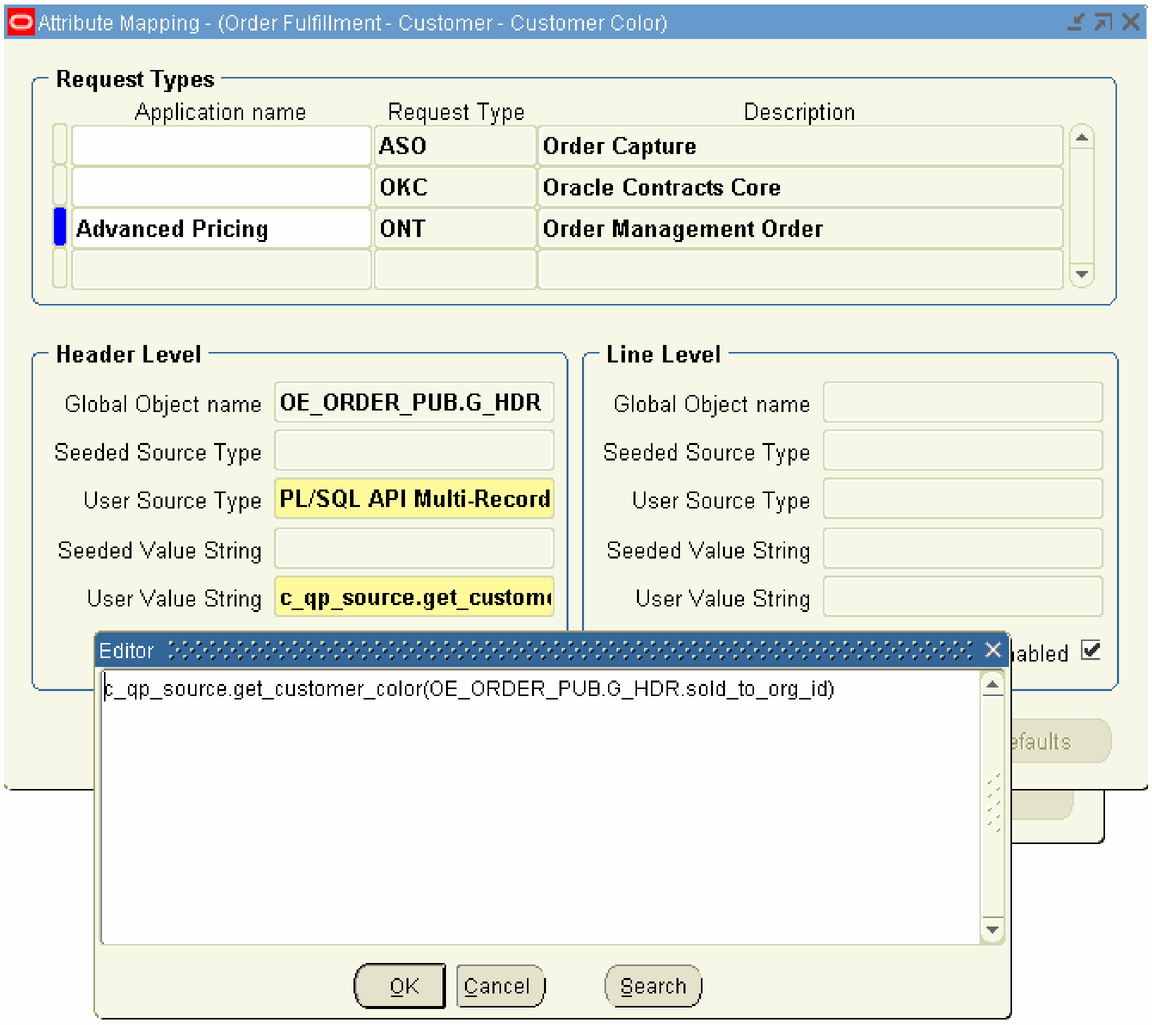


Link Attributes - (Order Fulfillment) (Qualifier Context - Customer)

Code	Name	Precedence	Level	Attribute Mapping Method	LOV Enabled	Use
INVOICE_TO_PA	Invoice To Party Site	400	BOTH	ATTRIBUTE MAPPING	✓	
PARTY_ID	Party ID	360	BOTH	ATTRIBUTE MAPPING	✓	
SALES_CHANNE	Sales Channel	320	BOTH	ATTRIBUTE MAPPING	✓	
SHIP_TO	Ship To	250	BOTH	ATTRIBUTE MAPPING	✓	
SHIP_TO_PART	Ship To Party Site	380	BOTH	ATTRIBUTE MAPPING	✓	
SITE_ORG_ID	Site Use	270	BOTH	ATTRIBUTE MAPPING	✓	
CUSTOMER_CO	Customer Color	260	ORDER	ATTRIBUTE MAPPING	✓	
SOLD_TO_ORG_	Customer Name	260	BOTH	ATTRIBUTE MAPPING	✓	

The Attribute Mapping button opens the following form where the actual PL/SQL API, PL/SQL multi-record logic, or, Profile Option is linked to the Attribute. The global structure defined by the request type can be referenced in PL/SQL syntax. In this case, a customer may have more than one color, therefore a PL/SQL API Multi-Record structure will be returned by the custom function.

Figure 7



Attribute Mapping - (Order Fulfillment - Customer - Customer Color)

Request Types

Application name	Request Type	Description
	ASO	Order Capture
	OKC	Oracle Contracts Core
Advanced Pricing	ONT	Order Management Order

Header Level

Global Object name	OE_ORDER_PUB.G_HDR
Seeded Source Type	
User Source Type	PL/SQL API Multi-Record
Seeded Value String	
User Value String	c_qp_source.get_custom

Line Level

Global Object name	
Seeded Source Type	
User Source Type	
Seeded Value String	
User Value String	

Editor

```
c_qp_source.get_customer_color(OE_ORDER_PUB.G_HDR.sold_to_org_id)
```

After the attribute link has been saved, the concurrent program Build Attribute Mapping Rules must be executed.

Before executing Build Attribute Mapping Rules, the PL/SQL package / function: *c_qp_source.get_customer_color* must be developed and built in the database. The following is the package / function signature:

```
CREATE OR REPLACE PACKAGE C_QP_SOURCE AS
/*
|| Package for custom attribute mapping
*/
FUNCTION get_customer_color
            (p_sold_to_org_id      IN    NUMBER)
RETURN QP_ATTR_MAPPING_PUB.t_multirecord;

END C_QP_SOURCE;
```

The package body will contain PL/SQL logic to obtain Customer Colors from a data source that may / or may not be a core Oracle Applications table.

The Build Attribute Mapping Rules concurrent program reads all PL/SQL linked attributes and drops and re-creates the QP_BUILD_SOURCING _PVT package. This package simply builds calls to both seeded and custom sourcing rules. The concurrent program accomplishes this in two steps. First it builds a QP_BUILD_SOURCING_PVT_TMP package and attempts to create this in the database. If any compilation errors occur, then the concurrent program exits with an error and does not attempt to build QP_BUILD_SOURCING_PVT. Execute the concurrent report: Attribute Mapping Rules Error Report for a list of compilation errors. If QP_BUILD_SOURCING _PVT_TMP is successfully compiled, then the QP_BUILD_SOURCING _PVT package is dropped and re-created using the new mapping rules.

The code that results loops through each enabled qualifier, pricing, and product attribute and executes that function. The following code snippet displays a portion of QP_BUILD_SOURCING_PVT that is built by Build Attribute Mapping Rules. Both seeded calls for CUSTOMER, QUALIFIER_ATTRIBUTE2, Customer Name (as displayed in Figure 4), and the custom CUSTOMER, QUALIFIER_ATTRIBUTE35, Customer Color are shown.

```
…

--  Src_Type: API
    BEGIN
      v_attr_value :=
                      OE_ORDER_PUB.G_LINE.sold_to_org_id;
    EXCEPTION
      WHEN OTHERS THEN
        v_attr_value := NULL;
    END;

    BEGIN
    IF v_attr_value = FND_API.G_MISS_NUM THEN
      v_attr_value := NULL;
    END IF;
    EXCEPTION
      WHEN VALUE_ERROR THEN
        IF v_attr_value = FND_API.G_MISS_CHAR THEN
          v_attr_value := NULL;
        END IF;
      WHEN OTHERS THEN
        v_attr_value := NULL;
    END;

    IF (v_attr_value IS NOT NULL) THEN

      x_qual_ctxts_result_tbl(q_count).context_name := 'CUSTOMER';
      x_qual_ctxts_result_tbl(q_count).attribute_name := 'QUALIFIER_ATTRIBUTE2';
      x_qual_ctxts_result_tbl(q_count).attribute_value := v_attr_value;
      q_count := q_count + 1;

        IF l_debug = FND_API.G_TRUE THEN
          oe_debug_pub.add('After product assigned');
        END IF;
    END IF;--v_attr_(m)value
--  Src_Type: API_MULTIREC
    BEGIN
      v_attr_mvalue :=
                 c_qp_source.get_customer_color(OE_ORDER_PUB.G_HDR.sold_to_org_id);
    EXCEPTION
      WHEN OTHERS THEN
        IF l_debug = FND_API.G_TRUE THEN
          oe_debug_pub.add('Multirec API error');
        END IF;
    END;

    IF (v_attr_mvalue.count <> 0) AND (v_attr_mvalue(1) IS NOT NULL) THEN
      v_index := 1;
      LOOP
        x_qual_ctxts_result_tbl(q_count).context_name := 'CUSTOMER';
        x_qual_ctxts_result_tbl(q_count).attribute_name := 'QUALIFIER_ATTRIBUTE35';
        x_qual_ctxts_result_tbl(q_count).attribute_value := v_attr_mvalue(v_index);
        q_count := q_count + 1;
        IF l_debug = FND_API.G_TRUE THEN
          oe_debug_pub.add('After product assigned');
        END IF;

        EXIT WHEN v_index = v_attr_mvalue.LAST;
        v_index := v_index + 1;
      END LOOP;
    END IF;--v_attr_(m)value
…
```

This code is executed *by QP_ATTR_MAPPING_PUB.Build_Contexts* at the time a transaction is priced. This procedure returns two (2) PL/SQL tables: one that holds all QUALIFIER / ATTRIBUTE / VALUE combinations, the other, all PRODUCT and PRICING /ATTRIBUTE / VALUE combinations. This information is passed to the pricing engine to calculate a price.

Best Practices

- Use a common PL/SQL Package for all functions used for custom attribute linking.
- Use engine debug in custom code to assist in troubleshooting mapping errors. These statements will appear in a debug file when Help-> Diagnostics-> Debug is turned on for a session.

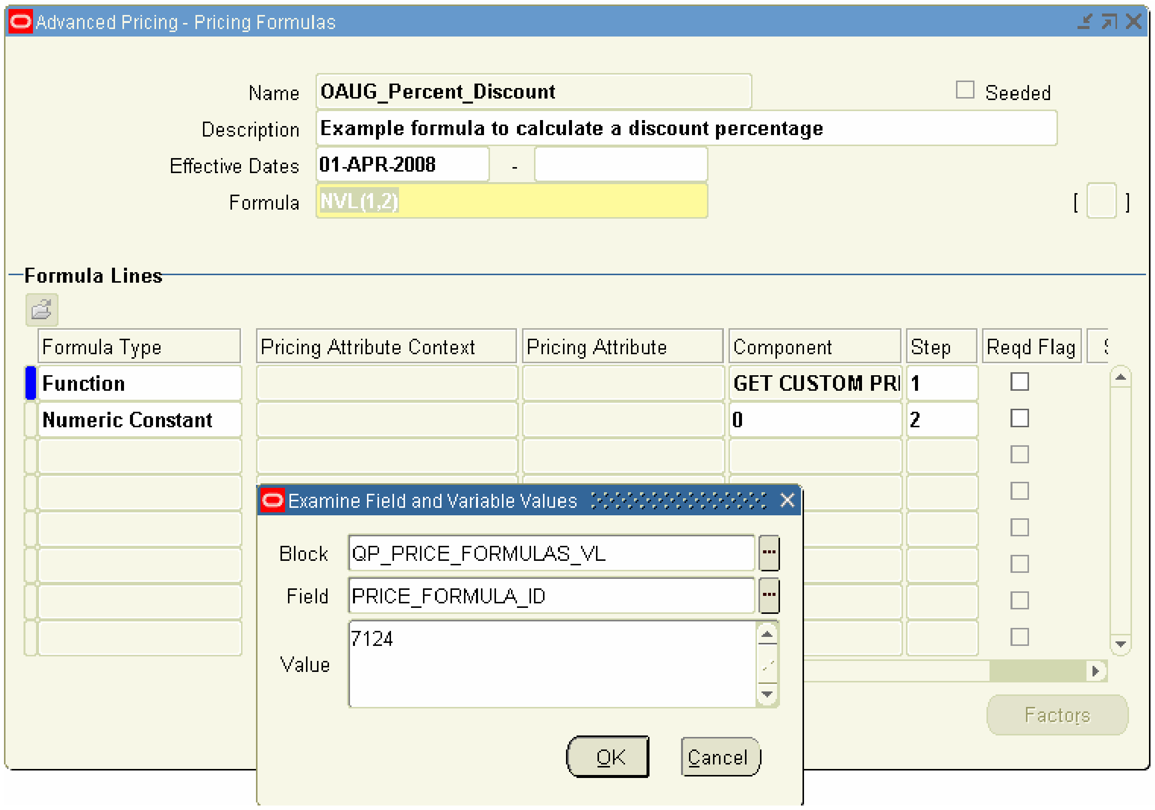
```
IF qp_preq_grp.G_DEBUG_ENGINE = FND_API.G_TRUE THEN
   qp_preq_grp.engine_debug('=> c_qp_custom.get_custom_color: ');
   qp_preq_grp.engine_debug('=> Parameters : ');
   qp_preq_grp.engine_debug('=> p_sold_to_org_id :'||p_ship_to_org_id);
END IF;
```

- Execute QP Build Sourcing Rules during system down time, or quiet system times. Objects need to be locked for this process to complete successfully.

Formulas: Get Custom Price

QP_CUSTOM.Get_Custom_Price is an Oracle Applications provided PL/SQL package function used to capture logic to derive a value that can be substituted as a formula line step value, list price (dynamic or static), or a modifier line (lump sum, percent, new amount). The custom logic used to derive a value can use any value copied to the global PL/SQL structures of the request system.

Figure 8


Advanced Pricing - Pricing Formulas

Name: OAUG_Percent_Discount
Seeded: ☐
Description: Example formula to calculate a discount percentage
Effective Dates: 01-APR-2008 -
Formula: NVL(1,2)

Formula Lines

Formula Type	Pricing Attribute Context	Pricing Attribute	Component	Step	Reqd Flag
Function			GET CUSTOM PRI	1	☐
Numeric Constant			0	2	☐

Examine Field and Variable Values

Block: QP_PRICE_FORMULAS_VL
Field: PRICE_FORMULA_ID
Value: 7124

To define a Formula, navigate to the Formula Setup screen. Define the formula name, description, effective dates, and, Formula; the arithmetic statement referencing formula lines as steps. To extend to custom PL/SQL code, enter the Formula Type as ‘Function’ with a component of GET_CUSTOM_PRICE. After the Formula is saved, obtain the Price_Formula_ID from the Help -> Diagnostic -> Examine feature. The Price_Formula_ID is passed to GET_CUSTOM_PRICE when the Pricing Engine determines that a value from this formula is required to satisfy a pricing request.

In the QP_CUSTOM package, the GET_CUSTOM_PRICE signature is defined as:

```
FUNCTION Get_Custom_Price
(
 p_price_formula_id     IN NUMBER,
 p_list_price           IN NUMBER,
 p_price_effective_date IN DATE,
 p_req_line_attrs_tbl   IN QP_FORMULA_PRICE_CALC_PVT.REQ_LINE_ATTRS_TBL
)
RETURN NUMBER;
```

Modify the function body to include the PL/SQL logic. The logic should be accesses by:

```
c_OAUG_Formula_ID := 7124; -- Formula ID corresponding to OAUG_Example
```

There are two (2) methods to obtain the parameter values needed for a discount percent in this example:

1. Obtain from the p_req_line_attrs_tbl parameter. Remember, only Pricing and Product Attributes are available in the `p_req_line_attrs_tbl` parameter. Therefore, the sold_to_ord_id (customer) must be obtained using method 2.

```
  FOR i IN 1..p_req_line_attrs_tbl.count
  LOOP
    -- Assign Inventory Item ID
    If (p_req_line_attrs_tbl(i).attribute_type = 'PRODUCT')   and
       (p_req_line_attrs_tbl(i).context        = 'ITEM')      and
       (p_req_line_attrs_tbl(i).attribute      = ’PRICING_ATTRIBUTE1')
    Then
        l_inventory_item_id := p_req_line_attrs_tbl(i).value;
    End If;

  END LOOP;
```

2. Obtain from the request type’s global structure.

```
l_sold_to_org_id := OE_ORDER_PUB.G_HDR.sold_to_ord_id;
```

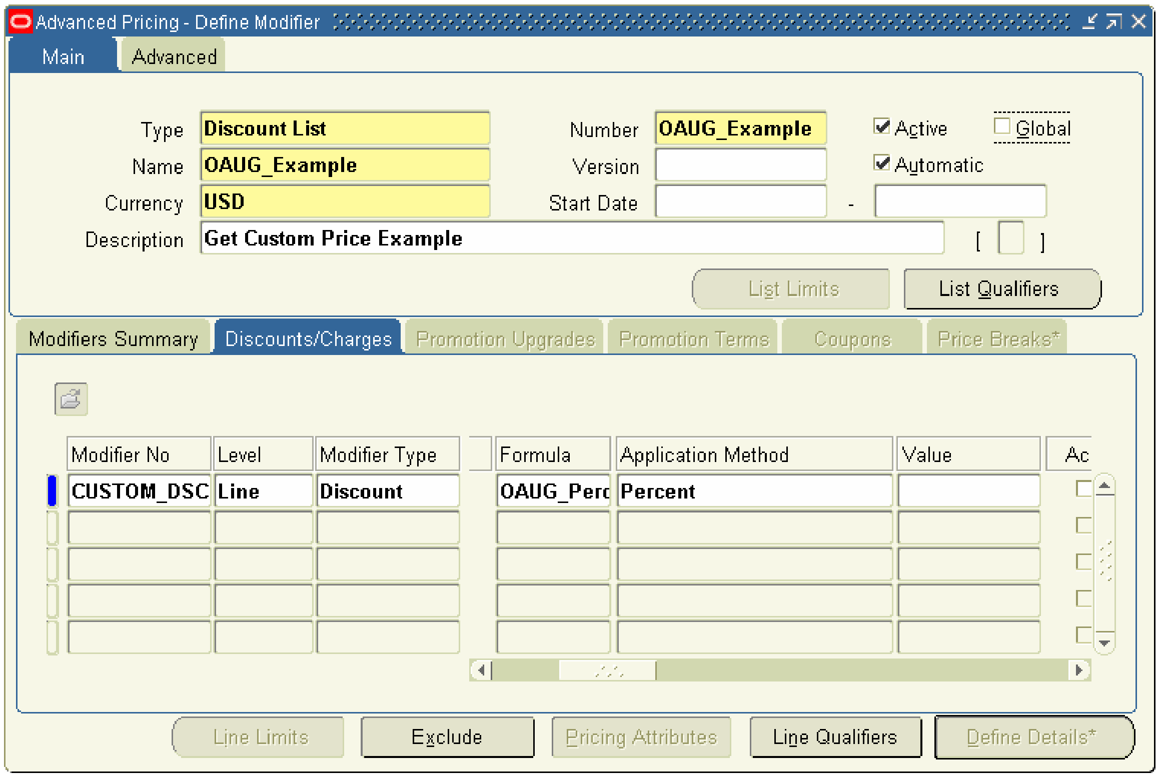
The values from the order transaction can now be used to calculate the discount percent.

```
IF p_price_formula_id = c_OAUG_Formula_ID THEN
     l_disc_percent := c_formula.discount_percent
                              (l_sold_to_org_id
                              ,l_inventory_item_id );

      RETURN l_disc_percent;
END IF;
```

Now attach the formula to a Modifier Line.

Figure 9


Advanced Pricing - Define Modifier

Main | Advanced

Type	Discount List	Number	OAUG_Example	✓ Active ☐ Global
Name	OAUG_Example	Version		✓ Automatic
Currency	USD	Start Date		-
Description	Get Custom Price Example			

List Limits | List Qualifiers

Modifiers Summary | Discounts/Charges | Promotion Upgrades | Promotion Terms | Coupons | Price Breaks*

Modifier No	Level	Modifier Type	Formula	Application Method	Value	Ac
CUSTOM_DSC	Line	Discount	OAUG_Perc	Percent		☐

Line Limits | Exclude | Pricing Attributes | Line Qualifiers | Define Details*

When the OAUG_Example modifier is qualified for this discount, the Formula OAUG_Percent_Discount will execute using custom logic to derive the discount percent.

Best Practices

- Use a common PL/SQL package for all functions custom pricing functions.
- Modularize the PL/SQL code outside GET_CUSTOM_PRICE. GET_CUSTOM_PRICE simply becomes a wrapper to the custom function. The custom function can then be reference outside a request for price.

Custom Request Types and Request Price

The *QP_PREQ_PUB.Price_Request* is Advanced Pricings public API that calls the pricing engine to obtain list price, selling price, as well as, pricing benefits, charges, and discounts that can be applied to a transaction. Request Price is the entry point to the pricing engine. It can be accessed by custom applications requiring a simple list price and discounted selling price, or, it can provide functionality to extend custom application's complete transaction pricing requirements. If the calling custom application captures adjustments and benefits related to a priced transaction, consider defining a new request type. This provides the greatest flexibility in capturing pricing adjustment lines related to the pricing request, as well as, the Global Structure specific to the custom application, used to pass information from the custom application to the pricing engine.

The following steps outline the components needed to make a call to *QP_PREQ_PUB .Price_Request:*

1. Copy the transaction information to the PL/SQL global structure defined by the Request Type. For the ONT (Order Management Orders) request type, these structures are OE_ORDER_PUB.G_HDR and OE_ORDER_PUB.G_LINE. Both structures contain most all columns associated with the OE_ORDER_HEADERS_ALL and OE_ORDER_LINES_ALL order management tables that store transaction information for an order.
2. For each transaction line, execute the *QP_ATTR_MAPPING_PUB.Build_Contexts* API to resolve the product, pricing, and qualifier CONTEXT / ATTRIBUTE VALUES.
3. Copy the values returned by Build_Contexts, and the transaction header and line information, to the parameters of *QP_PREQ_PUB.Price_Request*. Also, define the control record. Three values are required for the Control Record:
    a. Pricing Event. This defines the pricing phases that the Pricing Engine to consider when evaluating the request. Valid Pricing Events are:
        i. PRICE – List Line Base Price (List Price)
        ii. LINE – Includes the pricing phases of List Line Base Price, List Line Adjustment, Line Charges, and, Line Charges Manual.
        iii. BOOK – Includes the Book Event Phases only.
        iv. ORDER – All Lines Adjustments, Header Level adjustments and Header Level Charges
        v. SHIP – Includes Phases related to the Shipping Events
        vi. BATCH – All ORDER and LINE phases. This is used to capture all pricing for an order transaction.

    b. Calculate Flag – Determines the scope of work the pricing engine is being asked to perform:
        i. Search Only - Obtain a list price only. Do not calculate a selling price.
        ii. Calculate Only - Pass this if adjustment records need to be interpreted by the Pricing Engine. New adjustments are not retrieved. Typically, this is set when the Pricing Engine has been called, and modifications to adjustments have been made.
        iii. Search and Calculate - Used for the standard Pricing Engine call. New adjustments are captured and applied to the list price to obtain a new selling price.

    c. Simulation Flag – The Pricing Engine keeps track of issued and redeemed coupons. A value of ‘Y’es indicates that the call to pricing is a simulation and that the pricing engine should not make permanent record changes for issuing and redeeming coupons.

4. Call *QP_PREQ_PUB.Price_Request*.

If this call is made by a separate request type, logic must be developed to capture the information returned by *QP_PREQ_PUB.Price_Request*. The requirements of the interface will determine what needs to be captured and retained from the call.

Customizations

Customizations are a double edged sword. While providing your business users with functionality that enables productivity, customer satisfaction, and competitive advantage, introduced is a cost of development, maintenance, support and upgrade vulnerability.

Many times Oracle Application modules provide a way to meet the custom requirement; however, the implementation of the requirement may not be straight forward and be cumbersome. The cost to implement with “vanilla” Oracle may out weigh the cost to develop a customization.

When to Customize

Evaluate the cost of doing business without the customization. Quantify and answer the following questions:

- Does the customization reduce the time to execute a change to meet changing market place requirements?
- Does the customization minimize data errors during maintenance periods?
- Does the customization provide a competitive advantage?
- Does the customization enable your business the bandwidth to develop and explore new opportunities?

Then evaluate the cost of the customization.

- What is the cost to develop and support?
- What is the cost and risk to patch and upgrade?

Customizing should be discouraged if it is of little business value.

How to Customize

If Customizations are developed correctly, the vulnerability to patches and upgrades can be minimized and avoided. Oracle Applications Developer Guide discusses two general methods of customization:

> **Customization by Modification**
> This method of customization involves modifying existing Oracle Application components to meet the custom requirement. This may mean changing a form, PL/SQL logic, or, tables that are provided by Oracle Applications. These types of modifications may be lost during the application of patches or upgrades, and, can introduce data integrity issues for standard Oracle processes.

Customization by Extension
Customization by extension involves developing PL/SQL, forms, database tables as stand alone entities that can be integrated with core Oracle Application modules. Using development methods documented in the Oracle Applications Developers Guide, this method can result custom modules that look and feel the same as the core application modules. Oracle Applications Architecture provides interface points such as Zoom Forms and Custom PLLs (form libraries) that allow standard form modules to interface with custom forms, as well as, APIs that support application logic to insure data integrity when maintaining data in standard oracle data structures.

Custom Data

Data accessed by the Advanced Pricing module should be in the same Oracle instance that Oracle Applications is running. If data sources exist outside the Oracle instance, consider replication or import methods to make the data available within the Oracle Application instance. Data referenced outside the Oracle Applications instance will result in performance issues.

When modeling custom data sources that are accessed by extending the Advanced Pricing module, be sure to design physical tables in a way that program logic can efficiently resolve a value from a data element on the transaction being priced. Information pertaining to a priced transaction typically is derived from a Customer or Item related data element. In a custom tables, consider referencing HZ_PARTIES.PARTY_ID, HZ_CUST_ACCOUNT.CUST_ACCOUNT_ID, and, HZ_CUST_SITE_USES_ALL.SITE_USE_ID for quick retrieval of customer related information, and, MTL_SYSTEM_ITEMS.INVENTORY_ITEM_ID and Item Categories for item related information. Logic to resolve values from custom data sources must be proficient to avoid performance issues.

About the author:

Chris Herdic as worked with Oracle Applications for more than 12 years. His first assignment on an Oracle Applications project was to develop a Custom Pricing Module for a 10.6 implementation. When this implementation upgraded to R11i, he lead the technical team in implementing one of the first successful deployments of the Advanced Pricing module. He is currently a senior Oracle Applications consultant at The Gates Corporation, and is a senior consulting member of OappsNet.