APTTUS®

CPQ on Salesforce Winter 2019 SOAP API Guide

12/04/2019

Table of Contents

About this Guide

Apttus CPQ on Salesforce SOAP API Guide explains the SOAP APIs provided by Apttus Configuration, Pricing, and Quoting (CPQ).

Topic	Description
What's Covered	This guide walks the API developers through the list of SOAP APIs provided by Apttus.
Primary Audience	• API develpoers
IT Environment	Refer to the latest *CPQ on Salesforce Release Notes* for information on System Requirements and Supported Platforms.
Updates	For a comprehensive list of updates to this guide for each release, see the W hat's New topic.
Other Resources	• *CPQ on Salesforce Administrator Guide*: Refer to this guide for information on configuring Apttus CPQ. • *CPQ on Salesforce User Guide*: Refer to this guide for information on features and use cases of Apttus CPQ. • *CPQ on Salesforce Release Notes*: Refer to this document for information on system requirements and supported platforms, new features and enhancements, resolved issues, and known issues for a specific release.

This guide describes the following topics:

- CPQ Web Service
- Quote/Proposal Config Web Service
- Collaboration Structure

Before using CPQ, you must be familiar with the following:

- Basic knowledge of Salesforce
- Basic knowledge of SOAP APIs
- Salesforce and Apttus terms and definitions

If you are new to Apttus CPQ, begin here: Get Started and About Apttus Configure Price Quote.

What's New

Release	Topic	Description
Winter 2019	Registering Split Cart Parameters	New topic. New API
	Batch Job Service	New topic.
	Splitting a Smart Cart	New topic. New API
	Updating Price for the Smart Cart	New topic. New API
Summer 2019	No updates	No new APIs are introduced in this release.
Spring 2019	Accepting a Quote	New topic. New API
	Adding Products to a Collaboration Request	New topic. New global method.
Winter 2018	No updates	No new APIs are introduced in this release. The guide is updated to reflect product name changes.

About Apttus Configure Price Quote

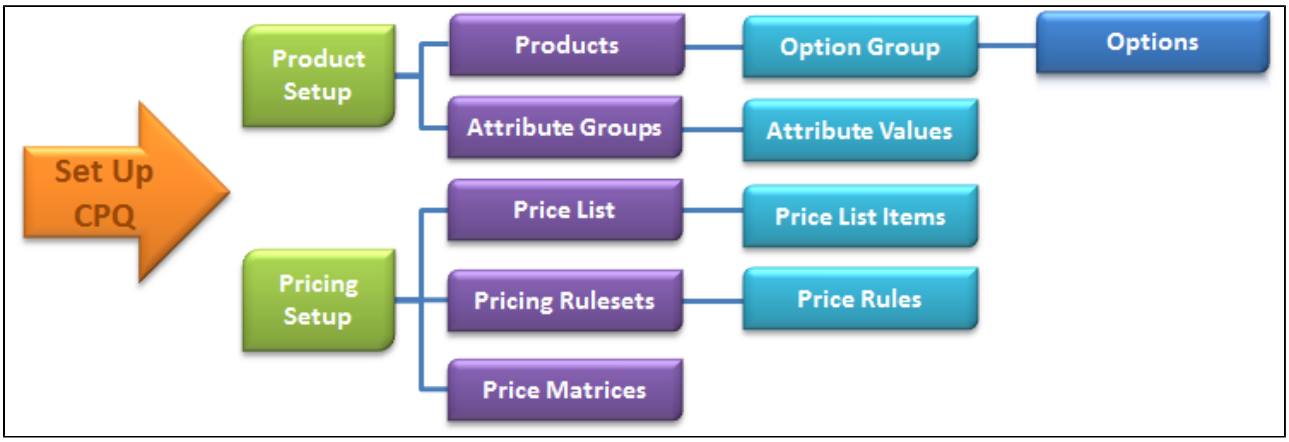
Apttus CPQ is designed to guide users to the best possible quotes for all products, pricing, and channels. In addition, by integrating quoting software with your contract processes, you can automate renewal quotes based on previously agreed upon pricing and terms.

Apttus CPQ gives you visibility and control of the quote-to-order phase of your sales process. Beginning in an opportunity, you can select products or services, configure and price them for simple or highly complex scenarios, and produce high-quality customer facing quotes and proposals. A proposal consists of configured products, a summary of the proposal, the price list associated with it, pricing for the configured products, associated opportunity, and shipping information.

Apttus CPQ is a web-based, on-demand application that is accessed through a standard web browser through the Internet.

Apttus CPQ Model

It is important to understand Apttus product and pricing objects and their relationships to each other in the overall application. Apttus CPQ is divided into two major areas - **Product Setup** and **Pricing Setup**. Products and Pricing are defined and customized based on how your organization wants to set up their configuration.



Getting Started

- API Supported Packages
  This guide covers CPQ and Configuration and Pricing APIs.
- Document Setup
  The Apttus CPQ and API reference Guide is divided into two sections: *API Reference* and *Scenarios*.
- API Standards and Development Platforms
  Apttus APIs are based on Salesforce APIs and use the same standards and platforms.
- Field Types
  Apttus APIs use a subset of the supported data and field types on Salesforce.

API Supported Packages

Refer to the *CPQ on Salesforce Release Notes* for the package details of the latest release.

Document Setup

The Apttus CPQ and CM API reference Guide is divided into two sections: *API Reference* and *Scenarios*.

API Reference	The *API reference* section details the APIs that you can use to manipulate Apttus objects through API calls and passing parameters. The API reference section also includes some code samples.
Scenarios	The *Scenarios* section details examples of the APIs you require to complete a specific task such as, adding products and constraint rules to the cart. The scenarios are classified by theme. You can refer to the generic examples of scenarios to identify the calls you can use to achieve your objective.

API Standards and Development Platforms

Apttus APIs are based on Salesforce APIs and use the same standards and platforms

Standards

Name	Reference
Simple Object Access Protocol (SOAP) 1.1	http://www.w3.org/TR/2000/NOTE-SOAP-20000508
Web Service Description Language (WSDL) 1.1	http://www.w3.org/TR/2001/NOTE-wsdl-20010315
WS-I Basic Profile 1.1	http://www.ws-i.org/Profiles/BasicProfile-1.1-2004-08-24.html

Development Platforms

Apttus SOAP API works with standard SOAP development environments. For a list of compatible development platforms, see Salesforce Developer Force API details.

Field Types

Apttus APIs use a subset of the supported data and field types on Salesforce.

The following table lists the APIs that Apttus provides. For a comprehensive list of all field types supported by Salesforce, see Salesforce Data Types.

Type	Description
Boolean	The **Boolean** field has atrue(or 1) orfalse(or 0) value.
Data object	The **Data Object** field is an ID type and is represented byCPQ.nnDOin this document.

Type	Description
Date	The **Date** field contains date values only and do not contain relevant time values. Time in a date field is always set to midnight in the UTC time zone. If you want a timestamp you must use a **dateTime** field.
Decimal	The **Decimal** field provides an exact numeric value and you can arbitrarily size the precision and scale of the value.
ID	The **ID** field is an alphanumeric field that acts like the primary key for a specific record associated with an object. The ID value includes a three-character code that identifies which object the record is associated with. The ID for a specific record does not change. For some objects, this field may also be areferencetype value, which contains the ID value for a related record. They are identified by field names ending in 'Id', such aspriceListId. The ID field acts like foreign keys and their values can be changed using anupdate()call.
Integer	The **Integer** field contains whole numbers only. There are no digits after the decimal.
List	The **List** field includes a fixed set of values from which you must select a single value. Picklists are available as drop-down lists. If a picklist is unrestricted, the API does not limit entries to only currently active values.
String	The **String** field contains text and may have differing length restrictions based on the data you store in the specific field. For instance,Citymay be limited to 50 characters, whileAddressLine1is limited to 255 characters.

Recommendations

Following are the recommendations for CPQ SOAP APIs on the Salesforce Platform:

- You must do the CPQ master data integration by using the public Admin APIs.
  In a nutshell, the master data includes product, rules, and pricing setup as a whole.

- You cannot use generic APIs for any of the master datasets.
  Note that Apttus CPQ does not provide support for any data, which is being integrated through the generic API mechanism for customer implememtation.

API Reference

In this section:

CPQ Web Service

The CPQ web service APIs account for the standard actions to configure, price, and quote.

You can use the CPQ web service APIs to complete the following tasks:

- Creating a Favorite Configuration
  This API creates a cart for the quote or proposal referenced by QuoteID. The quote or proposal must be associated with a price list.
- Retrieving Categories for a Price List
  This API is used to get the unique list of categories for all the products within the specified price list.
- Retrieving Products and List Prices for a Price List
  This API is used to get the list of all the products within the specified price list.
- Retrieving Products and List Prices for a Price List and Category
  This API is used to get the list of products and related list prices for a specified category in the price list.
- Retrieving Products and List Prices For a Price List and Search Text
  This API is used to get the list of products that match search text criteria you used for the products in the price list.
- Retrieving Products and List Prices For a Price List Category and Search Text
  This API is used to get the list of products that match search text criteria you used and belong to specified category id in the price list.

- Retrieving Option Groups, Options, and List Prices for a Price List Product
  This retrieves option groups, options, and list prices for a price list product.
- Modifying Assets (Deprecated)
  This method enables you to complete standard asset based ordering actions of renew, terminate, amend, upgrade and increment.
- Retrieving Asset Line Items
  Gets the list of asset line item objects matching the parameters used as the search criteria.
- Comparing Products
  This is used to retrieve products, based on product IDs, and compare them side by side. This is typically done from the Product Catalog, to decide between products before adding them to the cart.
- Adding a Bundle to a Cart
  This adds a bundle along with selected products, options, and updated quantity to the cart.
- Adding Options to a Bundle
  This adds one or more Options Products to a Bundle product.
- Adding Products to a Cart
  This API adds one or more products (with default options) to the cart along with quantity, term, start date, and end date.
- Adding Price Ramps to a Cart (CPQ Web Service)
  After you add a line item to a cart, this API enables you to add primary and secondary ramp line items for the line item. Once the ramp line items are created, you can also update the ramp line item details or delete ramp line item details using standard SOQL queries.
- Associating Constraint Rules to a Cart
  This associates constraints rules to a cart.
- Applying Constraint Rules to a Cart
  This processes and applies all the constraint rules.
- Removing a Bundle from a Cart
  This removes a product bundle and related line items, options from the cart.
- Removing Multiple Bundles from a Cart
  This removes one or more products (with default options) or bundles from the cart along with quantity, term, start date, and end date.
- Applying Constraint Rules to Deleted Products
  This runs the rules that are related to the deleted products.

- Retrieving Constraint Rules Results
  This retrieves the constraints rules applied to a cart.
- Computing the Net Price for a Bundle
  This API is used to calculate the price for the individual and summary line items of the cart. It runs the pricing rules, calculate bundle, line item, option level net prices and update the cart line with calculated prices. After the pricing is calculated for the cart line items, the resulting information is stored in the *Line Item* and *Summary Group* objects.
- Finalizing a Cart's Contents
  This finalizes the cart, synchronizing the cart line items with the quote/proposal.
- Synchronizing a Cart
  This is used to sync the shopping cart with the product configuration on the quote record.
- Abandoning a Cart
  This deletes the selected cart in **Draft** status.
- Updating Quote Terms
  This method enables you to update the expected start date and expected end date of the quote, along with its selling term. This method does the following:
- Price Breakup for a Cart or Specific Line Item
  This method can be used to retrieve the price breakup for a cart or specific line item.
- Updating Price For A Cart
  This method enables you to update the price for items in a given cart. Only line items in pending status are updated.
- Computing Shipping for Cart Line Items
  This API enables you to calculate the shipping amount for an entire order and does not display the breakup for each line item.
- Computing Taxes for Cart Line Items
  This API enables you to calculate tax breakups for cart line item.
- CPQAdmin Web Service
  The APIs in this Webservice allow you to create products, categories, hierarchies, and attribute groups. Once you have created the items, you can also choose to retrieve them using the APIs provided in the Webservice.

Creating a Cart from a Quote

This API creates a cart for the quote or proposal referenced by QuoteID. The quote or proposal must be associated with a price list.

createCart

Parameters			
Name	**Type**	**Required?**	**Description**
request	CPQ.CreateCartRequestDO	Yes	The request data object.

Request Data Object - CPQ.CreateCartRequestDO		
Field	**Type**	**Description**
QuoteID	ID	The id of the quote/proposal to be associated with the cart.
Properties	List<Apttus_Config2.Property>	The list of properties applicable to the cart

Data Object - Config2.Property		
Field	**Type**	**Description**
Name	String	Specify the features applicable to the cart. Applicable values are: • useAdvancedApproval: Enables Advanced Approval for a cart. • useDealOptimizer: Enables Deal Optimizer for a cart
Value	String	The applicable values are true or false. Specifying the value as true enables the feature for a cart.

Response Data Object - CPQ.CreateCartResponseDO		
Field	**Type**	**Description**
CartId	ID	The ID of the newly created cart object

Code Sample

The sample below enables you to create a cart for a valid quote with a Quote ID. Using the sample below, you can search for a valid quote using a quote number. If a quote exists with the quote number entered, you can create a cart using the createCart API or you will be prompted with a message to enter a valid quote number. You can invoke this API in use cases when you want to show a cart page based on the quote. For example for a realized opportunity, you can create a quote. Based on a valid quote ID, you can create a cart using this API.

```
/**
 * The below method demonstrates how to create a cart for a quote
 */
public static void createCart(String quoteNumber) {

    List<Apttus_Proposal__Proposal__c> quote = [SELECT Id FROM
Apttus_Proposal__Proposal__c WHERE Name = :quoteNumber LIMIT 1];

    if(!quote.isEmpty()) {

        // Create the request object
        Apttus_CPQApi.CPQ.CreateCartRequestDO request = new
Apttus_CPQApi.CPQ.CreateCartRequestDO();
        request.QuoteId = quote.get(0).Id;

        // Excute the createCart routine
        Apttus_CPQApi.CPQ.CreateCartResponseDO response =
Apttus_CPQApi.CPQWebService.createCart(request);

        System.debug('Cart has been successfully created. CartId = '
+ response.CartId);
    }
}
```

Retrieving Categories for a Price List

This API is used to get the unique list of categories for all the products within the specified price list.

getCategoriesForPriceList

Parameters			
Name	**Type**	**Required?**	**Description**
priceListId	ID	Yes	The id of the price list

Response Data Object - CPQ.CategorySearchResultDO		
Field	**Type**	**Description**
HasCategories	Boolean	Indicates if there are categories for the price list.
Categories	List<CPQ.CategoryDO>	The list of category data objects.

Data Object - CPQ.CategoryDO		
Field	**Type**	**Description**
CategoryId	ID	The Id of the category.
Name	String	The category name.
ParentCategoryId	ID	The Id of the parent category.
HasChildCategories	Boolean	Indicates if the category has child categories.
ChildCategories	List<CPQ. CategoryDO>	The list of child category data objects associated with the category.

Data Object - CPQ.CategoryDO		
Field	**Type**	**Description**
ProductCount	Integer	The number of products within a category.

Code Sample

The code sample below enables you to search for categories based on the pricelist ID. All the categories and subcategories associated with the price list are displayed. You can search for categories associated with a price list till the nth level. You can provide a search field on the cart page that enables the user to search for products by categories. For example, your cart page has multiple categories, such as Laptop, camera, Desktop, Accessories and so on. The user should be able to search for products by category by entering category name such as laptop. Use this API to invoke the categories and its associated sub-categories of products and then display it to the user.

```
public void executeSearch()
{
//Query for fetching pricelist id by querying price list
name
     List<Apttus_Config2__PriceList__c> priceListItemList = [select
id
     from Apttus_Config2__PriceList__c
     where  name = :priceListName limit 1];

//If an id is returned in the list execute the
getCategoriesforpricelist API.
      if(priceListItemList.size() > 0)
     {

           priceListId = priceListItemList[0].ID;

            Apttus_CPQApi.CPQ.CategorySearchResultDO result =
Apttus_CPQApi.CPQWebService.getCategoriesForPriceList(priceListId);

            lstwrap = New List<CategoryWrapperClass>();
             For( Apttus_CPQApi.CPQ.CategoryDO catresult : result.
Categories)
               {
                  CategoryWrapperClass wrap = New
CategoryWrapperClass();
                  wrap.CategoryId = catresult.CategoryId;
```

```
                    wrap.categoryName = catresult.name;
                    lstwrap.add(wrap);
  //If a category has sub categories fetch the sub-category name and
id
            if(catresult.HasChildCategories)
                {
                    For( Apttus_CPQApi.CPQ.CategoryDO subcatresult :
catresult.ChildCategories)
                     {
                       CategoryWrapperClass subwrap = New
CategoryWrapperClass();
                        subwrap.CategoryId = subcatresult.CategoryId;
                        subwrap.categoryName = subcatresult.name;
                        lstwrap.add(subwrap);
   // If the sub category has child categories fetch the sub
category name and
ID.
      if(subcatresult.HasChildCategories)
    {
       For( Apttus_CPQApi.CPQ.CategoryDO subsubcatresult :
subcatresult.ChildCategories)
         {
             CategoryWrapperClass subsubwrap = New
CategoryWrapperClass();
             subsubwrap.CategoryId = subsubcatresult.CategoryId;
             subsubwrap.categoryName = subsubcatresult.name;
             lstwrap.add(subsubwrap);

}
                    }
                }
             }
           }
         }
//If no Price List exists with the searched string name execute the
else condition
 else
  {

  lstwrap = New List<CategoryWrapperClass>();
   ApexPages.addMessage(new ApexPages.Message(ApexPages.severity.
info, 'Category Name not found. Please enter valid Category Name.'));
     }
}
```

Retrieving Products and List Prices for a Price List

This API is used to get the list of all the products within the specified price list.

This API automatically takes into consideration product visibility rules and will enforce them if applicable. For more information on Product Visibility, see the *CPQ Admin Guide*.

getProductsForPriceList

Parameters		
Name	**Type**	**Description**
priceListId	ID	The id of the price list.

Response Data Object - CPQ.ProductSearchResultDO		
Field	**Type**	**Description**
Products	List<CPQ.ProductDO>	The list of product data objects.
HasProducts	Boolean	This returns true if the list of product data objects is not empty.

Data Object - CPQ.ProductDO		
Field	**Type**	**Description**
ProductId	ID	The Id of the product.
ProductCode	String	The product code.
Name	String	The product name.
Description	String	The product description.

Data Object - CPQ.ProductDO		
Field	**Type**	**Description**
ImageUrl	String	The location of the image, if there is one, associated with the product.
ContentUrl	String	The product content location.
HasPrices	Boolean	Indicates if there are list prices for the product.
HasProperties	Boolean	Indicates if there are properties set for the product.
Property	List<CPQ. PropertyDO>	The list of property data objects.
Prices	List<CPQ. PriceDO>	The list of price data objects.

Data Object - CPQ.PropertyDO		
Field	**Type**	**Description**
Name	String	The name of the property.
Value	Decimal	The value of a property.

Data Object - CPQ.PriceDO		
Field	**Type**	**Description**
ChargeType	String	The charge type.
Value	Decimal	The list price.

Data Object - CPQ.PriceDO		
Field	**Type**	**Description**
Apttus_Config2__PriceListItem__c PriceItem	Apttus Object	The price list items for a price list.

Code Sample

Using the sample you can show the user the list of products for a particular price list. For example if you have a price list created for a particular company, the user can search the price list by name and view all the products associated with that price list. For example, when a user searches for badger price list and clicks search, invoke the getProductsForPriceList API in which you pass the pricelistID as a parameter. Fetch and then display all the product components to the user such as Machinery, standard price, quantity and name and description of the product.

```
public void getProductsForPriceList()
{
//If the priceList name by which the user searches the price list
exists fetch the ID.
    if(priceListId == null || priceListId== '0')
    {
        List<Apttus_Config2__PriceList__c> priceListItemList =
[select id
             from Apttus_Config2__PriceList__c
             where  Name = :priceListName limit 1];

//If the priceListItem exists, the list size >0 assign the ID at the
0th position of the list to priceListId
        if(priceListItemList.size() > 0)
        {
            priceListId = priceListItemList[0].ID;
        }
        else
        {
            lstProductwrapAll = New List<ProductWrapperClass>();
            lstProductwrap = New List<ProductWrapperClass>();
            return;
        }
    }

//Fetch id
```

```
Apttus_CPQApi.CPQ.ProductSearchResultDO productResult =
        Apttus_CPQApi.CPQWebService.getProductsForPriceList
(priceListId);

    productCount = 'Product Count: ' + productResult.Products.size();

    lstProductwrapAll = New List<ProductWrapperClass>();
    lstProductwrap = New List<ProductWrapperClass>();

//For the fetched pricelistID fetch and display the following
For(Apttus_CPQApi.CPQ.ProductDO catresult : productResult.Products)
    {
        ProductWrapperClass wrap = New ProductWrapperClass ();
        wrap.ProductId = catresult.ProductId;
        wrap.ProductCode= catresult.ProductCode;
        wrap.ProductName=catresult.Name;
        wrap.Description=catresult.Description;
        wrap.ImageUrl=catresult.ImageUrl;
        wrap.ContentUrl=catresult.ContentUrl;
        wrap.HasPrices=catresult.HasPrices;
        wrap.Prices=catresult.Prices;
        wrap.Quantity=1;
        lstProductwrapAll.add(wrap);
    }

}
```

Retrieving Products and List Prices for a Price List and Category

This API is used to get the list of products and related list prices for a specified category in the price list.

This API automatically takes into consideration product visibility rules and will enforce them if applicable. For more information on Product Visibility, see the *CPQ Admin Guide*.

getProductsForPriceListCategory

Parameters		
Name	**Type**	**Description**
priceListId	ID	The id of the price list.

Parameters		
Name	**Type**	**Description**
categoryID	ID	The id of the category.

Response Data Object - CPQ.ProductSearchResultDO		
Field	**Type**	**Description**
Products	List<CPQ.ProductDO>	The list of product data objects.
HasProducts	Boolean	This returns true if the list of product data objects is not empty.

Data Object - CPQ.ProductDO		
Field	**Type**	**Description**
ProductId	ID	The Id of the product.
ProductCode	String	The product code.
Name	String	The product name.
Description	String	The product description.
ImageUrl	String	The location of the image, if there is one, associated with the product.
ContentUrl	String	The product content location.
HasPrices	Boolean	Indicates if there are list prices for the product.

Data Object - CPQ.ProductDO		
Field	**Type**	**Description**
HasProperties	Boolean	Indicates if there are properties set for the product.
Property	List<CPQ.PropertyDO>	The list of property data objects.
Prices	List<CPQ.PriceDO>	The list of price data objects.

Data Object - CPQ.PropertyDO		
Field	**Type**	**Description**
Name	String	The name of the property.
Value	Decimal	The value of a property.

Data Object - CPQ.PriceDO		
Field	**Type**	**Description**
ChargeType	String	The charge type.
Value	Decimal	The list price.
Apttus_Config2__PriceListItem__c PriceItem	Apttus Object	The price list items for a price list.

Code Sample

The sample below enables you to search for categories using price list name. Once the user selects the category and proceeds to search, pass the pricelistID and categoryID as parameters to the API. The user can select and view the products associated with that category and it

components. For example, user enters the price list name and selects the category-Hardware. All the products associated with the Hardware category, such as Laptop are displayed. You can display all the fields associated to that product.

```
public void getProductList()
{
    categoryId ='';

    for(CategoryWrapperClass wrap: lstwrap )
    {
        if(wrap.selected == true)
        {
            categoryId = categoryId + wrap.CategoryId +',';
        }
    }
//If no category is selected prompt the user with an appropriate
error message.
    if(categoryId.Trim()=='')
    {
        ApexPages.addMessage(new ApexPages.Message(ApexPages.
severity.info, 'Please select at least one Category.'));
        lstProductwrap = New List<ProductWrapperClass>();
        lstProductwrapAll = New List<ProductWrapperClass>();
    }
//Pass the pricelistID and categoryID as parameters to the API
    else
    {
        Apttus_CPQApi.CPQ.ProductSearchResultDO productResult =
     Apttus_CPQApi.CPQWebService.getProductsForPriceListCategory
(priceListId,categoryId.Substring(0,categoryId.length()-1));

        productCount = 'Product Count: ' + productResult.Products.
size();

        lstProductwrap = New List<ProductWrapperClass>();
        lstProductwrapAll = New List<ProductWrapperClass>();
        For(Apttus_CPQApi.CPQ.ProductDO catresult : productResult.
Products)
        //Fetch and display the following product fields to the user
{
            ProductWrapperClass wrap = New ProductWrapperClass ();
            wrap.ProductId = catresult.ProductId;
            wrap.ProductCode= catresult.ProductCode;
            wrap.ProductName=catresult.Name;
            wrap.Description=catresult.Description;
```

```
            wrap.ImageUrl=catresult.ImageUrl;
            wrap.ContentUrl=catresult.ContentUrl;
            wrap.HasPrices=catresult.HasPrices;
            wrap.Prices=catresult.Prices;
            wrap.Quantity=1;
            lstProductwrapAll.add(wrap);
        }

    }
}
```

Retrieving Products and List Prices For a Price List and Search Text

This API is used to get the list of products that match search text criteria you used for the products in the price list.

This API automatically takes into consideration product visibility rules and will enforce them if applicable. For more information on Product Visibility, see the *CPQ Admin Guide*.

getProductsForSearchText

Parameters		
Name	**Type**	**Description**
priceListId	ID	The id of the price list.
searchText	String	The search terms that will be used to retrieve the products for the price list.

Response Data Object - CPQ.ProductSearchResultDO		
Field	**Type**	**Description**
Products	List<CPQ.ProductDO>	The list of product data objects.
HasProducts	Boolean	This returns true if the list of product data objects is not empty.

Data Object - CPQ.ProductDO		
Field	**Type**	**Description**
ProductId	ID	The Id of the product.
ProductCode	String	The product code.
Name	String	The product name.
Description	String	The product description.
ImageUrl	String	The location of the image, if there is one, associated with the product.
ContentUrl	String	The product content location.
HasPrices	Boolean	Indicates if there are list prices for the product.
HasProperties	Boolean	Indicates if there are properties set for the product.
Property	List<CPQ.PropertyDO>	The list of property data objects.
Prices	List<CPQ.PriceDO>	The list of price data objects.

Data Object - CPQ.PropertyDO		
Field	**Type**	**Description**
Name	String	The name of the property.
Value	String	The value of a property.

Data Object - CPQ.PriceDO		
Field	**Type**	**Description**
ChargeType	String	The charge type.
Value	Decimal	The list price.
Apttus_Config2__PriceListItem__c PriceItem	Apttus Object	The price list items for a price list.

Code Sample

Using the sample below you can search for products using the price list id and product name search string. For example, you can provide to search fields for the user one for price list and the other for Search Text. Once the user enters the price list name, fetch the price list ID using the SOQL query. The user then types in the Product Name and clicks Search. Invoke the API where you pass the priceListID and the corresponding search string as parameters. The response object returns the product object fields. For example, if the user enters the name of the Hardware price list and the name of the product, such as Laptop and clicks Search, invoke the API and pass the priceListId and the product name as parameters of the API and fetch and display the product information.

```
class ProductWrapperClass
{
    Id ProductId;
    String ProductCode;
    String ProductName;
    String Description;
    String ImageUrl;
    String ContentUrl;
    Boolean HasPrices;
    List<Apttus_CPQApi.CPQ.PriceDO> Prices;
}

public void getProductsForSearchText(Id priceListId, String
productName) {
```

```
    //Pass the pricelist ID and the Product Name entered in the
search text as parameters to the API

    Apttus_CPQApi.CPQ.ProductSearchResultDO productResult =
Apttus_CPQApi.CPQWebService.getProductsForSearchText(priceListId,
productName);

    String productCount = 'Product Count: ' + productResult.Products.
size();

    List<ProductWrapperClass> lstProductwrapAll = New
List<ProductWrapperClass>();


    For(Apttus_CPQApi.CPQ.ProductDO catresult : productResult.
Products)

    {
        ProductWrapperClass wrap = New ProductWrapperClass();
        wrap.ProductId = catresult.ProductId;
        wrap.ProductCode= catresult.ProductCode;
        wrap.ProductName=catresult.Name;
        wrap.Description=catresult.Description;
        wrap.ImageUrl=catresult.ImageUrl;
        wrap.ContentUrl=catresult.ContentUrl;
        wrap.HasPrices=catresult.HasPrices;
        wrap.Prices=catresult.Prices;
        lstProductwrapAll.add(wrap);
    }
    ApexPages.addMessage(new ApexPages.Message(ApexPages.severity.
info, 'Product Count : ' + productCount));

    ApexPages.addMessage(new ApexPages.Message(ApexPages.severity.
info, 'Product Data Object : ' + lstProductwrapAll));

}
```

Retrieving Products and List Prices For a Price List Category and Search Text

This API is used to get the list of products that match search text criteria you used and belong to specified category id in the price list.

This API automatically takes into consideration product visibility rules and will enforce them if applicable. For more information on Product Visibility, see the *CPQ Admin Guide*.

getProductsForCategorySearchText

Parameters		
Name	**Type**	**Description**
priceListId	ID	The id of the price list.
categoryId	ID	The id of the category.
searchText	String	The search terms that will be used to retrieve the products for the price list.

Response Data Object - CPQ.ProductSearchResultDO		
Field	**Type**	**Description**
Products	List<CPQ.ProductDO>	The list of product data objects.
HasProducts	Boolean	This returns true if the list of product data objects is not empty.

Data Object - CPQ.ProductDO		
Field	**Type**	**Description**
ProductId	ID	The Id of the product.
ProductCode	String	The product code.
Name	String	The product name.
Description	String	The product description.

ImageUrl	String	The location of the image, if there is one, associated with the product.
ContentUrl	String	The product content location.
HasPrices	Boolean	Indicates if there are list prices for the product.
HasProperties	Boolean	Indicates if there are properties set for the product.
Property	List<CPQ.PropertyDO>	The list of property data objects.
Prices	List<CPQ.PriceDO>	The list of price data objects.

Data Object - CPQ.PropertyDO		
Field	**Type**	**Description**
Name	String	The name of the property.
Value	Decimal	The value of a property.

Data Object - CPQ.PriceDO		
Field	**Type**	**Description**
ChargeType	String	The charge type.
Value	Decimal	The list price.

Data Object - CPQ.PriceDO		
Field	**Type**	**Description**
Apttus_Config2__PriceListItem__c PriceItem	Apttus Object	The price list items for a price list.

Code Sample

The user can enter the name of price list to view all the categories belonging to that price list. The sample below defines the behavior when the user selects a category and searches a product from the categories displayed. Fetch and pass the priceListID, categoryID, and productID as parameters to the API and display the fields of CPQ.ProductDo object. For example, the user enters a valid price list name, all the subsequent categories for the pricelist are displayed. The user selects the category Hardware and searches for a product Laptop. All the products matching the search string laptop are displayed along with the fields such as price, name , id and so on. You can execute the count loop to specify the number of products to be displayed per page and to provide next and previous pages.

```
public void getProductsForCategorySearchText()
{
//Search for single category multiple category using comma separator
and fetch category id
    categoryId ='';

    for(CategoryWrapperClass wrap: lstwrap )
    {
        if(wrap.selected == true)
        {
            categoryId = categoryId + wrap.CategoryId +',';
        }
    }
//If no category selected is selected by the user, show the
following message
    if(categoryId.Trim()=='')
    {
        ApexPages.addMessage(new ApexPages.Message(ApexPages.
severity.info, 'Please select at least one Category.'));
        lstProductwrap = New List<ProductWrapperClass>();
        lstProductwrapAll = New List<ProductWrapperClass>();
    }
```

```
//If searched price list is valid, categories for that pricelist are
displayed and if the user selects a category and enters search text
by 'productName' parameter, pass the PricelistID, categoryID, and
productID as parameters.
    else
    {
        Apttus_CPQApi.CPQ.ProductSearchResultDO productResult =
Apttus_CPQApi.CPQWebService.getProductsForCategorySearchText
(priceListId,categoryId.Substring(0,categoryId.length()-1),
productName);

        productCount = 'Product Count: ' + productResult.Products.
size();

        lstProductwrap = New List<ProductWrapperClass>();
        lstProductwrapAll = New List<ProductWrapperClass>();
        For(Apttus_CPQApi.CPQ.ProductDO catresult : productResult.
Products)
        {
// Fetch and display the following components for the products
            ProductWrapperClass wrap = New ProductWrapperClass ();
            wrap.ProductId = catresult.ProductId;
            wrap.ProductCode= catresult.ProductCode;
            wrap.ProductName=catresult.Name;
            wrap.Description=catresult.Description;
            wrap.ImageUrl=catresult.ImageUrl;
            wrap.ContentUrl=catresult.ContentUrl;
            wrap.HasPrices=catresult.HasPrices;
            wrap.Quantity=1;
            wrap.Prices=catresult.Prices;
            lstProductwrapAll.add(wrap);
        }

    }
}
```

Retrieving Option Groups, Options, and List Prices for a Price List Product

This retrieves option groups, options, and list prices for a price list product.

getOptionGroupsForPriceListProduct

Parameters		
Name	**Type**	**Description**
pricelistId	ID	The Id of the price list.
productId	ID	The Id of the product bundle.

Response Data Object - CPQ.ProductOptionGroupSearchResultDO		
Name	**Type**	**Description**
HasOptionGroups	Boolean	Indicates if the bundle has option groups.
OptionGroups	List<CPQ.ProductOptionGroupDO>	The list of product option group data objects.

Data Object - CPQ.ProductOptionGroupDO		
Field	**Type**	**Description**
ProductOptionGroupId	ID	Id of the product option group.
Name	String	The option group name.
Label	String	The option group label.
ParentOptiongroupId	ID	Id of the parent option group.
MinOptions	Integer	Minimum number of options that must be selected from the group.
MaxOptions	Integer	

Data Object - CPQ.ProductOptionGroupDO		
Field	**Type**	**Description**
		Maximum number of options that can be selected from the group.
HasChildOptionGroups	Boolean	Indicates whether there are child option groups.
ChildOptionGroups	List<CPQ. ProductOptionGroupDO>	List of child option group data objects.
HasOptionComponents	Boolean	Indicates if the option group has product components.
OptionComponents	List<CPQ.Product OptionComponentDO>	List of product component data objects.

Data Object - CPQ.ProductOptionComponentDO		
Field	**Type**	**Description**
ComponentId	ID	Id of the component.
ComponentProductId	ID	Id of the component product.
ProductCode	String	The product code.
Name	String	The product name.
Description	String	The product description.
ImageUrl	String	The location of the image, if there is one, associated with the product.

Data Object - CPQ.ProductOptionComponentDO		
Field	**Type**	**Description**
IsDefault	Boolean	Indicates if the option is included by default for the bundle.
IsRequired	Boolean	Indicates if the option is mandatory for the bundle.
HasPrices	Boolean	Indicates if there are list prices for the product.
Prices	List<CPQ.PriceDO>	List of price data objects.

Data Object - CPQ.PriceDO		
Field	**Type**	**Description**
ChargeType	String	The charge type.
Value	Decimal	The list price.
Apttus_Config2__PriceListItem__c PriceItem	Apttus Object	The price list items for a price list.

Code Sample

Using the sample below you enable the user to search and select products by a price list. You can then pass the IDs of the price list and product as arguments to the API. If in the result object HasOptionGroups returns true, fetch and display the option group components. If the option group components HasComponents=true display the option components such as quantity. For example if the user searches a product and clicks Search, invoke the API to fetch the option groups for that bundle.

```
public void getOptionGroupsForPriceListProduct()
{
```

```
    string productID='';
//Fetch product ID of selected products
    for(ProductWrapperClass selProdWrap: lstProductwrap)
    {
        if(selProdWrap.selected)
        {
            productID=selProdWrap.ProductId;
            break;
        }
    }

//If no product is selected, show an appropriate error message
    if(productID=='')
    {
        ApexPages.addMessage(new ApexPages.Message(ApexPages.
severity.info, 'Please select at least one Product.'));

        lstOptionGroup = new List<Apttus_CPQApi.CPQ.
ProductOptionGroupDO>();
        lstProductOptionWrapper = new
List<ProductOptionWrapperClass>();
    }
//If product is selected, pass pricelistID and productID as
parameters to the API
    else
    {
        Apttus_CPQApi.CPQ.ProductOptionGroupSearchResultDO result =
Apttus_CPQApi.CPQWebService.getOptionGroupsForPriceListProduct
(priceListId, productID);

        lstOptionGroup = new List<Apttus_CPQApi.CPQ.
ProductOptionGroupDO>();
        lstProductOptionWrapper = new
List<ProductOptionWrapperClass>();
//If the product has option groups, i,e the flag HasOptionGroups
returns true run the loop below fetching the components.
        if(result.HasOptionGroups)
        {
//For an option group fetch and display the following components
            For(Apttus_CPQApi.CPQ.ProductOptionGroupDO optionResult
: result.OptionGroups)
            {
            ProductOptionWrapperClass objProductOptionWrapperClass=ne
w ProductOptionWrapperClass();
            objProductOptionWrapperClass.
ProductOptionGroupId=optionResult.ProductOptionGroupId;
            objProductOptionWrapperClass.Name=optionResult.Name;
```

```
            objProductOptionWrapperClass.Label=optionResult.Label;
            objProductOptionWrapperClass.
ParentOptiongroupId=optionResult.ParentOptiongroupId;
            objProductOptionWrapperClass.MinOptions=optionResult.
MinOptions;
            objProductOptionWrapperClass.MaxOptions=optionResult.
MaxOptions;
            objProductOptionWrapperClass.
HasChildOptionGroups=optionResult.HasChildOptionGroups;
            objProductOptionWrapperClass.
ChildOptionGroups=optionResult.ChildOptionGroups;

            List<ProductQuantityOptionWrapperClass>
listOptionComponent= new List<ProductQuantityOptionWrapperClass>();

//For an option group with components such as min, max and quantity
fetch and display the editable components to the user
for(Apttus_CPQApi.CPQ.ProductOptionComponentDO optionComponent :
optionResult.OptionComponents)
        {
        ProductQuantityOptionWrapperClass
objProductQuantityOptionWrapperClass = new
ProductQuantityOptionWrapperClass ();
        objProductQuantityOptionWrapperClass.OptionComponent =
optionComponent;
        objProductQuantityOptionWrapperClass.Quantity = 1;

//Fetch and display the list price of the option
components
            List<Apttus_CPQApi.CPQ.PriceDO> priceList = new
List<Apttus_CPQApi.CPQ.PriceDO>();
            for(Apttus_CPQApi.CPQ.PriceDO price :  optionComponent.
Prices)
            {
                priceList.Add(price);
            }
            objProductQuantityOptionWrapperClass.prices = priceList ;
            listOptionComponent.Add
(objProductQuantityOptionWrapperClass);
                }
                objProductOptionWrapperClass.OptionComponents =
listOptionComponent;

                lstOptionGroup.Add(optionResult);

                lstProductOptionWrapper.Add
(objProductOptionWrapperClass);
```

```
            }

            ApexPages.addMessage(new ApexPages.Message(ApexPages.
severity.info, 'Option Count: ' + lstProductOptionWrapper.size()));
        }
    }
}
```

Modifying Assets (Deprecated)

This method enables you to complete standard asset based ordering actions of renew, terminate, amend, upgrade and increment.

This method creates the cart line item and associates it to the corresponding asset id. Actions can be performed on a single line item or a selected set of asset line items. Actions can also be performed on standalone product or bundle product.

When doing a renewal, you can use any of the standard adjustment types, including *% Uplift* which is used to pass an uplift percentage that is applied to the *base price* of the renewed line item, resulting in an updated *net renewal* price.

> **Note**
>
> *CartId*, *AssetLineItemId__c*, and *AssetActions* are required parameters, the use of other parameters depends on your business case.

modifyAsset

Parameters		
Name	**Type**	**Description**
Apttus_CPQApi.CPQ.ModifyAssetRequestDO	request	This is the request call made by the method.

Request Data Object - Apttus_CPQApi.CPQ.ModifyAssetRequestDO		
Field	**Type**	**Description**
CartId	ID	The Id of cart.
AssetActions	List<Apttus_CPQApi.CPQ.AssetActionDO>	The list of asset actions, which is either *Renew*, *Amend*, *Increment*, *Cancel*, *Upgrade*.

Response Data Object - Apttus_CPQApi.CPQ.ModifyAssetResponseDO		
Field	**Type**	**Description**
LineNumber	List<Decimal>	The asset line number.
AssetActions	List<Apttus_CPQApi.CPQ.AssetActionDO>	The list of asset actions, which is either *Renew*, *Amend*, *Increment*, *Cancel*, *Upgrade*.

Data Object - Apttus_CPQApi.CPQ.AssetActionDO		
Field	**Type**	**Description**
AssetAction	String	This is the specific asset action for the asset line item, referenced by AssetLineItemId.
AssetLineItemId	ID	This is the ID for the asset line item that you are going to renew, amend, increment, or cancel. This is mandatory.
Quantity	Decimal	The amount of assets.
SellingTerm	Decimal	The selling term for the asset.

Data Object - Apttus_CPQApi.CPQ.AssetActionDO		
Field	**Type**	**Description**
StartDate	Date	This is the start date for the asset action.
EndDate	Date	This is the end date for the asset action. If you are terminating an asset you only need an end date.
AdjustmentType	String	This is the type of adjustment that will be used. You can select from types such as % Discount, Discount Amount, and Price Override. You specific implementation may have more or less types.
AdjustmentAmount	Decimal	Enter the value (number) for the adjustment. How the value is interpreted is based on the Adjustment Type. For example, 10 with an Adjustment Type of *% Uplift* means the price will be cut in half, whereas that same value of 10 with an Adjustment Type of *Price Override* means the price will become 10.
Comments	String	Text string containing comments.
Message	String	Text string containing comments.
Pending	Boolean	This indicates whether an asset action is pending.
CustomData	Apttus_Config2__LineItem__c	This can be used to include custom fields you have added to the asset object.
Selling Term	Decimal	

Data Object - Apttus_CPQApi.CPQ.AssetActionDO		
Field	**Type**	**Description**
		This is the selling term of the asset line item that you want to renew, amend, or increment.

Code Sample

You can amend a list of standalone or bundle products. You can change the dates, quantity, or some options for a product using the Amend API. You can also cancel or change options in an Option Group using Amend.

The Sample below enables you to change amend the start and end date for selected assets of an account. The response returns the lineitem number of the asset to be updated. The assetAction status, the status of the asset is modified and updated.

When the customer clicks Amend, you can invoke the modifyAssets API and choose to amend the date, quantity.

```
public void Amend()
{
//Create an assetAction variable with value as amend
    string assetAction = 'Amend';

    List<Apttus_CPQApi.CPQ.AssetActionDO> assetActDOList = new
List<Apttus_CPQApi.CPQ.AssetActionDO>();

//If Asset is selected in the cart page, execute the loop
for(AssetWrapperClass objAssetWrapper : lstAssetWrapper)
    {
        if(objAssetWrapper.selected)
        {
            Apttus_CPQApi.CPQ.AssetActionDO assetActDO = new
Apttus_CPQApi.CPQ.AssetActionDO();
            assetActDO.AssetAction = assetAction;
            assetActDO.AssetLineItemId = objAssetWrapper.AssetId ;
//Change the end date to five months from start date
            assetActDO.EndDate = Date.today().addMonths(5);
//assetAction pending is true
            assetActDO.Pending = true;
            assetActDO.Quantity = objAssetWrapper.Quantity;
```

```
//Amend the start date to include today's date
            assetActDO.StartDate = Date.today().addMonths(0);

            List<String> customFields = new List<String>();
            customFields.add('Apttus_Config2__Comments__c');
            customFields.add('Apttus_Config2__PricingStatus__c');

            assetActDO.CustomFields = customFields;

            Apttus_Config2__LineItem__c liSO = new
Apttus_Config2__LineItem__c();
            liSO.Apttus_Config2__Comments__c = 'Comments Added my
Modify Asset API Code';
            liSO.Apttus_Config2__PricingStatus__c = 'Complete';
            assetActDO.CustomData = liSO;

            assetActDOList.add(assetActDO);
        }
    }

//Execute the API for a specific cart using CartID
    Apttus_CPQApi.CPQ.ModifyAssetRequestDO modifyRequest = new
Apttus_CPQApi.CPQ.ModifyAssetRequestDO();
    modifyRequest.CartId = cartID;
    modifyRequest.AssetActions = assetActDOList;

    Apttus_CPQApi.CPQ.ModifyAssetResponseDO modifyReponse =
Apttus_CPQApi.CPQWebservice.modifyAsset(modifyRequest);

    List<Decimal> lineNumberList = modifyReponse.LineNumbers;

    List<Apttus_CPQApi.CPQ.AssetActionDO> assetList = modifyReponse.
AssetActions;

//Signifies the status of the assetAction
    Boolean bIsPending = false;

    Integer iCounter = 0;

    while (iCounter < 1000)
    {
        bIsPending = false;
        for(Apttus_CPQApi.CPQ.AssetActionDO objAsset: assetList)
        {
            if(objAsset.Pending == true)
            {
                bIsPending = true;
```

```
                break;
            }
        }
//If the asset action is pending execute the loop below to change
the status
        if(bIsPending == true)
        {
            modifyReponse = Apttus_CPQApi.CPQWebservice.modifyAsset
(modifyRequest);
            lineNumberList = modifyReponse.LineNumbers;
            assetList = modifyReponse.AssetActions;
        }
        else
        {
            break;
        }
        iCounter++;
    }

    bindAssets();
}
//This function enables you to bind updated assets to an account
dynamically.
public void bindAssets()
{
    List<Apttus_Config2__AssetLineItem__c> aliSOList = [select id,
Apttus_Config2__Description__c,
    Apttus_Config2__Quantity__c, Apttus_Config2__StartDate__c,
Apttus_Config2__EndDate__c from Apttus_Config2__AssetLineItem__c
    where Apttus_Config2__AccountID__c = :accountID];

    lstAssetWrapper = new List<AssetWrapperClass>();

    for(Apttus_Config2__AssetLineItem__c aliSO : aliSOList)
    {
        AssetWrapperClass objAssetWrapperClass = new
AssetWrapperClass ();
        objAssetWrapperClass.AssetId = aliSO.ID;
        objAssetWrapperClass.ProductId = aliSO.
Apttus_Config2__Description__c;
        objAssetWrapperClass.Quantity = aliSO.
Apttus_Config2__Quantity__c;
        objAssetWrapperClass.StartDate = aliSO.
Apttus_Config2__StartDate__c;
        objAssetWrapperClass.EndDate = aliSO.
Apttus_Config2__EndDate__c;
```

```
            lstAssetWrapper.Add(objAssetWrapperClass);
        }
    }
```

You can renew a list of standalone or bundled products. You can choose to renew recurring items, overall bundle, and recurring options. You can also cancel or change the quantity, price, or an entire option. The sample below enables you to renew the assets. When the user selects an asset and clicks renew, invoke this API and use the sample below to extend the end date of the assets by 12 months.

```
public void ReNew()
{
//Declare assetAction variable with value as Renew
    string assetAction = 'Renew';

    List<Apttus_CPQApi.CPQ.AssetActionDO> assetActDOList = new
List<Apttus_CPQApi.CPQ.AssetActionDO>();

//For a selected asset you can execute the loop
    for(AssetWrapperClass objAssetWrapper : lstAssetWrapper)
    {
        if(objAssetWrapper.selected)
        {
            Apttus_CPQApi.CPQ.AssetActionDO assetActDO = new
Apttus_CPQApi.CPQ.AssetActionDO();
            assetActDO.AssetAction = assetAction;
            assetActDO.AssetLineItemId = objAssetWrapper.AssetId ;
            //assetActDO.EndDate = Date.today().addMonths(12);
//Enables you to set end date as 12 months from the existing end date
            assetActDO.EndDate = objAssetWrapper.EndDate.addMonths(12
);
            assetActDO.Pending = true;
            assetActDO.Quantity = objAssetWrapper.Quantity;
            //assetActDO.StartDate = Date.today().addMonths(0);
//Increment start date by One day from the existing start
date
            assetActDO.StartDate = objAssetWrapper.EndDate.addDays(1)
;

            List<String> customFields = new List<String>();
            customFields.add('Apttus_Config2__Comments__c');
            customFields.add('Apttus_Config2__PricingStatus__c');

            assetActDO.CustomFields = customFields;
```

```
//Fetch asset line item number
            Apttus_Config2__LineItem__c liSO = new
Apttus_Config2__LineItem__c();
            liSO.Apttus_Config2__Comments__c = 'Comments Added my
Modify Asset API Code';
            liSO.Apttus_Config2__PricingStatus__c = 'Complete';
            assetActDO.CustomData = liSO;

            assetActDOList.add(assetActDO);
        }
    }
//Invoke the modify assets API and specify the cart for which the
API is invoked.
    Apttus_CPQApi.CPQ.ModifyAssetRequestDO modifyRequest = new
Apttus_CPQApi.CPQ.ModifyAssetRequestDO();
    modifyRequest.CartId = cartID;
    modifyRequest.AssetActions = assetActDOList;

    Apttus_CPQApi.CPQ.ModifyAssetResponseDO modifyReponse =
Apttus_CPQApi.CPQWebservice.modifyAsset(modifyRequest);

    List<Decimal> lineNumberList = modifyReponse.LineNumbers;

    List<Apttus_CPQApi.CPQ.AssetActionDO> assetList = modifyReponse.
AssetActions;

    Boolean bIsPending = false;

    Integer iCounter = 0;

    while (iCounter < 1000)
    {
        bIsPending = false;
        for(Apttus_CPQApi.CPQ.AssetActionDO objAsset: assetList)
        {
            if(objAsset.Pending == true)
            {
                bIsPending = true;
                break;
            }
        }
//If asset Action is pending execute the following loop
        if(bIsPending == true)
        {
            modifyReponse = Apttus_CPQApi.CPQWebservice.modifyAsset
(modifyRequest);
            lineNumberList = modifyReponse.LineNumbers;
```

```
            assetList = modifyReponse.AssetActions;
        }
        else
        {
            break;
        }
        iCounter++;
    }

    bindAssets();
}

//This function enables you to bind updated asset.
public void bindAssets()
{
    List<Apttus_Config2__AssetLineItem__c> aliSOList = [select id,
Apttus_Config2__Description__c,
    Apttus_Config2__Quantity__c, Apttus_Config2__StartDate__c,
Apttus_Config2__EndDate__c from Apttus_Config2__AssetLineItem__c
    where Apttus_Config2__AccountID__c = :accountID];

    lstAssetWrapper = new List<AssetWrapperClass>();

    for(Apttus_Config2__AssetLineItem__c aliSO : aliSOList)
    {
        AssetWrapperClass objAssetWrapperClass = new
AssetWrapperClass ();
        objAssetWrapperClass.AssetId = aliSO.ID;
        objAssetWrapperClass.ProductId = aliSO.
Apttus_Config2__Description__c;
        objAssetWrapperClass.Quantity = aliSO.
Apttus_Config2__Quantity__c;
        objAssetWrapperClass.StartDate = aliSO.
Apttus_Config2__StartDate__c;
        objAssetWrapperClass.EndDate = aliSO.
Apttus_Config2__EndDate__c;

        lstAssetWrapper.Add(objAssetWrapperClass);
    }
}
```

You can increment the quantity or the dates of the asset items using the increment asset action. For example, the user selects an asset item and clicks Increment. You can use the sample below to update the quantity and the end date by five months and bind the updated assets to the account.

```
public void Increment()
{
    //Declare the assetAction variable with value as Increment
    string assetAction = 'Increment';

    List<Apttus_CPQApi.CPQ.AssetActionDO> assetActDOList = new
List<Apttus_CPQApi.CPQ.AssetActionDO>();

    for(AssetWrapperClass objAssetWrapper : lstAssetWrapper)
    {
        if(objAssetWrapper.selected)
        {
//For a selected asset increment the quantity by 5 and increment the
end date by 5.
            Apttus_CPQApi.CPQ.AssetActionDO assetActDO = new
Apttus_CPQApi.CPQ.AssetActionDO();
            assetActDO.AssetAction = assetAction;
            assetActDO.AssetLineItemId = objAssetWrapper.AssetId ;
            assetActDO.EndDate = Date.today().addMonths(5);
            assetActDO.Pending = true;
            assetActDO.Quantity = objAssetWrapper.Quantity + 5;
            assetActDO.StartDate = Date.today().addMonths(0);

            List<String> customFields = new List<String>();
            customFields.add('Apttus_Config2__Comments__c');
            customFields.add('Apttus_Config2__PricingStatus__c');

            assetActDO.CustomFields = customFields;
        //Line number of the asset items
            Apttus_Config2__LineItem__c liSO = new
Apttus_Config2__LineItem__c();
            liSO.Apttus_Config2__Comments__c = 'Comments Added my
Modify Asset API Code';
            liSO.Apttus_Config2__PricingStatus__c = 'Complete';
            assetActDO.CustomData = liSO;

            assetActDOList.add(assetActDO);
        }
    }
//Invoke the modifyAsset API to increment the quantity and end date
    Apttus_CPQApi.CPQ.ModifyAssetRequestDO modifyRequest = new
Apttus_CPQApi.CPQ.ModifyAssetRequestDO();
    modifyRequest.CartId = cartID;
    modifyRequest.AssetActions = assetActDOList;
```

```
    Apttus_CPQApi.CPQ.ModifyAssetResponseDO modifyReponse =
Apttus_CPQApi.CPQWebservice.modifyAsset(modifyRequest);

    List<Decimal> lineNumberList = modifyReponse.LineNumbers;

    List<Apttus_CPQApi.CPQ.AssetActionDO> assetList = modifyReponse.
AssetActions;

    Boolean bIsPending = false;

    Integer iCounter = 0;

    while (iCounter < 1000)
    {
        bIsPending = false;
        for(Apttus_CPQApi.CPQ.AssetActionDO objAsset: assetList)
        {
            if(objAsset.Pending == true)
            {
                bIsPending = true;
                break;
            }
        }
//If asset action is pending, execute this loop
        if(bIsPending == true)
        {
            modifyReponse = Apttus_CPQApi.CPQWebservice.modifyAsset
(modifyRequest);
            lineNumberList = modifyReponse.LineNumbers;
            assetList = modifyReponse.AssetActions;
        }
        else
        {
            break;
        }
        iCounter++;
    }

    bindAssets();
}

//Bind updated asset items
public void bindAssets()
{
    List<Apttus_Config2__AssetLineItem__c> aliSOList = [select id,
Apttus_Config2__Description__c,
```

```
    Apttus_Config2__Quantity__c, Apttus_Config2__StartDate__c,
Apttus_Config2__EndDate__c from Apttus_Config2__AssetLineItem__c
    where Apttus_Config2__AccountID__c = :accountID];

    lstAssetWrapper = new List<AssetWrapperClass>();

    for(Apttus_Config2__AssetLineItem__c aliSO : aliSOList)
    {
        AssetWrapperClass objAssetWrapperClass = new
AssetWrapperClass ();
        objAssetWrapperClass.AssetId = aliSO.ID;
        objAssetWrapperClass.ProductId = aliSO.
Apttus_Config2__Description__c;
        objAssetWrapperClass.Quantity = aliSO.
Apttus_Config2__Quantity__c;
        objAssetWrapperClass.StartDate = aliSO.
Apttus_Config2__StartDate__c;
        objAssetWrapperClass.EndDate = aliSO.
Apttus_Config2__EndDate__c;

        lstAssetWrapper.Add(objAssetWrapperClass);
    }
}
```

The terminate asset action enables you to terminate or cancel the selected asset items. The sample below enables you to terminate the asset and change its status by setting the end date as current date.

```
public void Terminate()
{
//Declaring variable assetAction and assigning it a value Cancel
    string assetAction = 'Cancel';

    List<Apttus_CPQApi.CPQ.AssetActionDO> assetActDOList = new
List<Apttus_CPQApi.CPQ.AssetActionDO>();

//To terminate a selected Asset, set the end date as today
    for(AssetWrapperClass objAssetWrapper : lstAssetWrapper)
    {
        if(objAssetWrapper.selected)
        {
            Apttus_CPQApi.CPQ.AssetActionDO assetActDO = new
Apttus_CPQApi.CPQ.AssetActionDO();
            assetActDO.AssetAction = assetAction;
            assetActDO.AssetLineItemId = objAssetWrapper.AssetId ;
```

```
            assetActDO.EndDate = Date.today().addMonths(0);
            assetActDO.Pending = true;
            assetActDO.Quantity = objAssetWrapper.Quantity;
            //assetActDO.StartDate = Date.today().addMonths(0);

            List<String> customFields = new List<String>();
            customFields.add('Apttus_Config2__Comments__c');
            customFields.add('Apttus_Config2__PricingStatus__c');

            assetActDO.CustomFields = customFields;

            Apttus_Config2__LineItem__c liSO = new
Apttus_Config2__LineItem__c();
            liSO.Apttus_Config2__Comments__c = 'Comments Added my
Modify Asset API Code';
            liSO.Apttus_Config2__PricingStatus__c = 'Complete';
            assetActDO.CustomData = liSO;

            assetActDOList.add(assetActDO);
        }
    }
//For a specific cart ID, invoke the API and modify the assets based
on the terminate function above.
    Apttus_CPQApi.CPQ.ModifyAssetRequestDO modifyRequest = new
Apttus_CPQApi.CPQ.ModifyAssetRequestDO();
    modifyRequest.CartId = cartID;
    modifyRequest.AssetActions = assetActDOList;

    Apttus_CPQApi.CPQ.ModifyAssetResponseDO modifyReponse =
Apttus_CPQApi.CPQWebservice.modifyAsset(modifyRequest);

    List<Decimal> lineNumberList = modifyReponse.LineNumbers;

    List<Apttus_CPQApi.CPQ.AssetActionDO> assetList = modifyReponse.
AssetActions;

    Boolean bIsPending = false;

    Integer iCounter = 0;

    while (iCounter < 1000)
    {
        bIsPending = false;
        for(Apttus_CPQApi.CPQ.AssetActionDO objAsset: assetList)
        {
            if(objAsset.Pending == true)
            {
```

```
                bIsPending = true;
                break;
            }
        }
//if asset action is pending execute the loop below
        if(bIsPending == true)
        {
            modifyReponse = Apttus_CPQApi.CPQWebservice.modifyAsset
(modifyRequest);
            lineNumberList = modifyReponse.LineNumbers;
            assetList = modifyReponse.AssetActions;
        }
        else
        {
            break;
        }
        iCounter++;
    }

    bindAssets();
}

//Use the function below to bind the updated asset items.
public void bindAssets()
{
    List<Apttus_Config2__AssetLineItem__c> aliSOList = [select id,
Apttus_Config2__Description__c,
    Apttus_Config2__Quantity__c, Apttus_Config2__StartDate__c,
Apttus_Config2__EndDate__c from Apttus_Config2__AssetLineItem__c
    where Apttus_Config2__AccountID__c = :accountID];

    lstAssetWrapper = new List<AssetWrapperClass>();

    for(Apttus_Config2__AssetLineItem__c aliSO : aliSOList)
    {
        AssetWrapperClass objAssetWrapperClass = new
AssetWrapperClass ();
        objAssetWrapperClass.AssetId = aliSO.ID;
        objAssetWrapperClass.ProductId = aliSO.
Apttus_Config2__Description__c;
        objAssetWrapperClass.Quantity = aliSO.
Apttus_Config2__Quantity__c;
        objAssetWrapperClass.StartDate = aliSO.
Apttus_Config2__StartDate__c;
        objAssetWrapperClass.EndDate = aliSO.
Apttus_Config2__EndDate__c;
```

```
            lstAssetWrapper.Add(objAssetWrapperClass);
        }
    }
```

The upgrade asset action enables you to update selected asset items. The sample below enables you to upgrade the asset. Using the sample below you can fetch the asset items using the getAssetsForSearchText API and upgrade the selected assetLineItems using AssetAction = Upgrade. You can upgrade the date and the quantity for an asset line item.

```
public static void processForUpgrade(ID cartId, ID accountId){
            //Declaration
            Set<ID> assetLineItemIds = new Set<ID>();

            Apttus_CPQApi.CPQ.AssetSearchResultDO result
             = Apttus_CPQApi.CPQWebService.getAssetsForSearchText
(accountId, null, null);

            Apttus_CPQApi.CPQ.ModifyAssetRequestDO assetRequest = new
 Apttus_CPQApi.CPQ.ModifyAssetRequestDO();
            assetRequest.CartId = cartId;


            for (Apttus_Config2__AssetLineItem__c assetItemSO :
result.AssetItems) {
                Apttus_CPQApi.CPQ.AssetActionDO assetAction = new
Apttus_CPQApi.CPQ.AssetActionDO();
                assetAction.AssetAction = 'Upgrade';
                assetAction.AssetLineItemId = assetItemSO.Id;
                assetAction.Quantity = assetItemSO.
Apttus_Config2__Quantity__c.intValue();
                assetAction.EndDate = Date.today();
                assetAction.Pending = true;
                assetLineItemIds.add(assetItemSO.Id);
                assetRequest.AssetActions.add
(assetAction);
            }

            Apttus_CPQApi.CPQ.ModifyAssetResponseDO assetResponse =
Apttus_CPQApi.CPQWebService.modifyAsset(assetRequest);

            Set<Decimal> lineNumbers = new Set<Decimal>
(assetResponse.LineNumbers);
```

```
            for (Apttus_Config2__LineItem__c lineSO : [Select Id,
Apttus_Config2__AssetLineItemId__c from Apttus_Config2__LineItem__c
Where Apttus_Config2__ConfigurationId__c = :cartId]){


            }
    }
```

Fetching Asset Line Items

This API fetches all Asset Line Items for various Accounts.

getAssetLineItems

Parameters		
Name	**Type**	**Description**
Apttus_Config2.CPQStruct. QueryAssetsRequestDO	request	Request object invoked by the method

Request Data Object - CPQStruct.QueryAssetsRequestDO		
Field	**Type**	**Description**
AccountIds	List <Id>	List of Account Ids

Response Data Object - Apttus_Config2.CPQStruct.QueryAssetsResponseDO		
Field	**Type**	**Description**
AssetLineItems	List<Apttus_Config2__AssetLineItem__c>	List of Asset Line Items.

Code Sample

The code sample below helps you get Asset Line Items for a list of Accounts. Use this API to fetch all asset line items associated with an account.

```
// create list of account ids
List<ID> listAccount = new List<ID>();
listAccount.add(accountId);

// create and populate request object
Apttus_Config2.CPQStruct.QueryAssetsRequestDO request =
    new Apttus_Config2.CPQStruct.QueryAssetsRequestDO();
request.AccountIds = listAccount;
request.FieldNames = null; // retrieve all fields in Asset Line Item
sObject
// sort by billing end date (optinal)
request.SortFields = new List<String>{'Apttus_Config2__BillingEndDate
__c'};
request.Descending = false; // sort ascending
request.CustomFilter = 'Apttus_Config2__BillingEndDate__c > 2011-04-
30'; // optional
// do not offset and do not limit number of records returned
request.Offset = null;
request.Nrecord = null;

// call getAssetLineItems API
Apttus_Config2.CPQStruct.QueryAssetsResponseDO response =
    Apttus_Config2.AssetService.getAssetLineItems(request);

List<Apttus_Config2__AssetLineItem__c> listItems = response.
AssetLineItems;
```

Changing Assets

You can invoke this API for changing Assets.

changeAssets

Parameters		
Name	**Type**	**Description**
Apttus_Config2.CPQStruct. ChangeAssetsRequestDO()	request	Request object invoked by the method

Request Data Object - CPQStruct.ChangeAssetsRequestDO()		
Name	**Type**	**Description**
AssetId	List<ID>	List of Asset IDs to change.
CartId	ID	The id of Cart for which an asset is being changed.

Response Data Object - Apttus_Config2.CPQStruct.ChangeAssetsResponseDO		
Field	**Type**	**Description**
Map <ID, LineItem__c>	LineItemMap	Returns all line items with all their field values.

Code Sample

The code sample below helps you make changes to an Asset by invoking this API.

```
// create list of asset ids
List<ID> listAssetId = new List<ID>();
for (AssetLineItemWrapperClass record : wrapperAssetLineItemList) {
    if (record.selected) {
        listAssetId.add(record.assetId);
    }
}

// create and populate request object
Apttus_Config2.CPQStruct.ChangeAssetsRequestDO request =
    new Apttus_Config2.CPQStruct.ChangeAssetsRequestDO();
request.AssetIds = listAssetId;
request.CartId = cartId;

// call changeAssets API
Apttus_Config2.CPQStruct.ChangeAssetsResponseDO response =
    Apttus_Config2.AssetService.changeAssets(request);

ApexPages.addMessage(new ApexPages.Message(ApexPages.severity.info,
                                           'changeAssets: ' +
response));
```

Renewing Assets

You can invoke this API to renew Assets.

RenewAssets

Parameters		
Name	**Type**	**Description**
Apttus_Config2.CPQStruct.RenewAssetsRequestDO()	request	Request object invoked by the method

Request Data Object - CPQStruct.RenewAssetsRequestDO		
Name	**Type**	**Description**
AssetIds	List<ID>	List of asset ids,
RenewEndDate	Date	The Asset End Date for Renewal
RenewTerm	Integer	The Renewal Term for an asset
CartId	ID	The id of Cart for which an asset is being renewed
FarthestAssetEndDate	Boolean	Setting this value to true renews assets using the farthest end date.

CPQStruct.RenewAssetsResponseDO		
Field	**Type**	**Description**
LineItemMap	Map<ID, Apttus_Config2__LineItem__c>	Returns all line items with all their field values.

Code Sample

The code sample below helps you renew Assets based on the specified Renewal Date or Renewal Term.

```
// create list of asset ids
List<ID> listAssetId = new List<ID>();
for (AssetLineItemWrapperClass record : wrapperAssetLineItemList) {
    if (record.selected) {
        listAssetId.add(record.assetId);
    }
}

// renew assets using an asset line item record with valid Renewal
Date and Term
if (objAssetLineItem.Apttus_Config2__RenewalDate__c != null
    || objAssetLineItem.Apttus_Config2__RenewalTerm__c != null) {

    // create and populate request object
    Apttus_Config2.CPQStruct.RenewAssetsRequestDO request =
        new Apttus_Config2.CPQStruct.RenewAssetsRequestDO();
    request.CartId = cartId;
    request.AssetIds = listAssetId;
    request.RenewEndDate = objAssetLineItem.
Apttus_Config2__RenewalDate__c;
    request.RenewTerm = objAssetLineItem.
Apttus_Config2__RenewalTerm__c;
    request.FarthestAssetEndDate = false;

    // call renewAssets API
    Apttus_Config2.CPQStruct.RenewAssetsResponseDO response =
        Apttus_Config2.AssetService.renewAssets(request);

    ApexPages.addMessage(new ApexPages.Message(ApexPages.severity.
info,
                                               'renewAssets: ' +
response));
```

Swapping Assets

You can invoke this API for swapping Assets.

swapAssets

Parameters		
Name	**Type**	**Description**
Apttus_Config2.CPQStruct.SwapAssetsRequestDO()	request	Request object invoked by the method

Request Data Object - Apttus_Config2.CPQStruct.SwapAssetsRequestDO()		
Name	**Type**	**Description**
ProductIds	List<ID>	Id of the product with which the Asset is being swapped.
NewStartDate	Date	The new Asset Start Date on Swapping
AssetIds	List<ID>	The Id of the Asset which is swapped.
CartIds	Id	The Id of Cart for which an Asset is being swapped

Response Data Object - Config2.AssetService.swapAssets()		
Field	**Type**	**Description**
LineItemMap	Map<ID, Apttus_Config2__LineItem__c>	Map of line items of the assets that are being swapped.

Code Sample

The code sample below helps you swap an Asset with another product.

```
// create list of asset ids
List<ID> listAssetId = new List<ID>();
for (AssetLineItemWrapperClass record : wrapperAssetLineItemList) {
    if (record.selected) {
        listAssetId.add(record.assetId);
```

```
    }
}

// create list of product ids
List<ID> listProductId = new List<ID>();
for (ProductWrapperClass product : wrapperProductList) {
    if (product.selected) {
        listProductId.add(product.productId);
    }
}

// create and populate request object
if (objAssetLineItem.Apttus_Config2__StartDate__c != null) {
    Apttus_Config2.CPQStruct.SwapAssetsRequestDO request =
        new Apttus_Config2.CPQStruct.SwapAssetsRequestDO();
    request.AssetIds = listAssetId;
    request.ProductIds = listProductId;
    request.NewStartDate = objAssetLineItem.
Apttus_Config2__StartDate__c;
    request.CartId = cartId;
}

// call swapAssets API
Apttus_Config2.CPQStruct.SwapAssetsResponseDO response =
    Apttus_Config2.AssetService.swapAssets(request);

ApexPages.addMessage(new ApexPages.Message(ApexPages.severity.info,
                                           'swapAssets: ' +
response));
```

Getting a list of Products to be swapped with Assets

getSwappedProducts

Fetches a list of products that are to be swapped.

> ⚠ Ensure that the Replacement Rule is active for Swapping Assets.

Parameters		
Name	**Type**	**Description**

Parameters		
Apttus_Config2.CPQStruct. RecommendationRequestDO()	request	Request object invoked by the method

Request Data Object - Apttus_Config2.CPQStruct.RecommendationRequestDO()		
Name	**Type**	**Description**
ProductIds	List<ID>	Id of the product with which the Asset is being swapped.
CartId	Id	Id of Cart for which Swap action is performed

Response Data Object - Apttus_Config2.CPQStruct.RecommendationResponseDO		
Field	**Type**	**Description**
ProductIds	List<ID>	Ids of products that can be swapped with assets.

Code Sample

The code sample below helps you fetch the list of Product Ids that can be swapped with Assets.

```
// create list of product ids
List<ID> listProductId = new List<ID>();
for (ProductWrapperClass product : wrapperProductList) {
    if (product.selected) {
        listProductId.add(product.productId);
    }
}

// create and populate request object
Apttus_Config2.CPQStruct.RecommendationRequestDO request =
    new Apttus_Config2.CPQStruct.RecommendationRequestDO();
request.cartId = cartId;
request.ProductIds = listProductId;

// call getSwappedProducts API
```

```
Apttus_Config2.CPQStruct.RecommendationResponseDO response =
    Apttus_Config2.AssetService.getSwappedProducts(request);

ApexPages.addMessage(new ApexPages.Message(ApexPages.severity.info,
                                           'getSwappedProducts: ' +
response));
```

Getting a count of Asset Line Items

This API gets the count of Asset Line Item in one or more accounts.

countAssetLineItems

Parameters		
Name	**Type**	**Description**
Apttus_Config2.CPQStruct. QueryAssetsRequestDO	request	Request object invoked by the method

Request Data Object - Apttus_Config2.CPQStruct.QueryAssetsRequestDO		
Field	**Type**	**Description**
AccountIds	List <Id>	List of Account Ids
CustomFilter	String	SOQL condition expression used as a custom filter. For example, user could request only assets after a given billing end date. This is optional.

Response Data Object - CPQStruct. QueryAssetsResponseDO		
Field	**Type**	**Description**
AssetCount	Integer	Number of Assets

Code Sample

The code sample below helps you get a count of Asset Line Items for a particular account. Use this API to fetch all asset line items associated with an account.

```
// create list of account ids
List<ID> listAccount = new List<ID>();
listAccount.add(accountId);

// create and populate request object
Apttus_Config2.CPQStruct.QueryAssetsRequestDO request =
    new Apttus_Config2.CPQStruct.QueryAssetsRequestDO();
request.AccountIds = listAccount;
request.CustomFilter = 'Apttus_Config2__BillingEndDate__c > 2011-04-
30'; // optional

// call countAssetLineItems API
Apttus_Config2.CPQStruct.QueryAssetsResponseDO response =
    Apttus_Config2.AssetService.countAssetLineItems(request);

ApexPages.addMessage(new ApexPages.Message(ApexPages.severity.info,
                                           'Asset Line Item Count: '
+ response.AssetCount));
```

Terminating Assets

You can invoke this API for Asset Cancellation.

cancelAssets

Parameters		
Name	**Type**	**Description**
Apttus_Config2.CPQStruct. CancelAssetsRequestDO()	request	Request object invoked by the method

Request Data Object - Apttus_Config2.CPQStruct.CancelAssetsRequestDO		
Name	**Type**	**Description**

Request Data Object - Apttus_Config2.CPQStruct.CancelAssetsRequestDO		
CancelDate	Date	The termination date of the Asset
AssetIds	List<ID>	List of asset ids to terminate.
CartId	ID	The id of Cart for which an asset is being canceled.

Response Data Object - Apttus_Config2.CPQStruct.CancelAssetsResponseDO		
Field	**Type**	**Description**
LineItemMap	Map <ID, Apttus_Config2__LineItem__c>	Line Items that are to be terminated.

Code Sample

The code sample below helps you terminate Assets based on the Termination Date.

```
// create list of asset ids
List<ID> listAssetId = new List<ID>();
for (AssetLineItemWrapperClass record : wrapperAssetLineItemList) {
    if (record.selected) {
        listAssetId.add(record.assetId);
    }
}

// create and populate request object
if (objAssetLineItem.Apttus_Config2__CancelledDate__c != null){
    Apttus_Config2.CPQStruct.CancelAssetsRequestDO request =
        new Apttus_Config2.CPQStruct.CancelAssetsRequestDO();
    request.CancelDate = objAssetLineItem.
Apttus_Config2__CancelledDate__c.date();
    request.AssetIds = listAssetId;
    request.CartId = cartId;
}

// call cancelAssets API
Apttus_Config2.CPQStruct.CancelAssetsResponseDO response =
    Apttus_Config2.AssetService.cancelAssets(request);
```

```
ApexPages.addMessage(new ApexPages.Message(ApexPages.severity.info,
                                           'cancelAssets: ' +
response));
```

Retrieving Asset Line Items

Gets the list of asset line item objects matching the parameters used as the search criteria.

getAssetsForSearchText

Parameters		
Name	**Type**	**Description**
accountId	ID	The id of the account associated with the asset. If you use this, you cannot pass locationId at the same time.
locationIds	List <ID>	The ids of the different locations associated with the assets. If you use this, you cannot pass accountId at the same time.
searchText	String	The search text.

Response Data Object - Apttus_CPQApi.CPQ.AssetSearchResultDO		
Name	**Type**	**Description**
HasAssetItems	Boolean	Indicates whether there are asset line items, that match the parameters used for the search.
AssetItems	List < Apttus_Config2__AssetLineItem__c >	This is a list of the asset line items. This returns the asset line item records and the values for the fields that belong to each record.

Code Sample

When you generate a quote for an existing customer, it is possible that the customer has some assets associated with the account. You can provide a search field on the cart page that enables the customer to search for existing assets and view them. When the user types the search criteria for fetching the assets and clicks Search, invoke the API. This sample should enable the user to search a product by product name, product code, product description, category name, and configuration type. The search for get assets is similar to the product search in the catalog page and installed products page.

```
public void searchAssets() {
    Apttus_CPQApi.CPQ.AssetSearchResultDO assetResults =
        Apttus_CPQApi.CPQWebService.getAssetsForSearchText
(accountId, null, assetSearchText);
    List<AssetWrapperClass> lstAssetWrapper = new
List<AssetWrapperClass>();

    // if assets items that are searched by a keyword exist (i.e
hasassetitems=true)
    // and some value is returned, show the following as a part of
the list
    if (assetResults.HasAssetItems == true) {
        List<Apttus_Config2__AssetLineItem__c> aliSOList =
assetResults.AssetItems;
        for (Apttus_Config2__AssetLineItem__c aliSO : aliSOList) {
            AssetWrapperClass objAssetWrapperClass = new
AssetWrapperClass ();
            objAssetWrapperClass.AssetId = aliSO.ID;
            objAssetWrapperClass.ProductId = aliSO.
Apttus_Config2__Description__c;
            objAssetWrapperClass.Quantity = aliSO.
Apttus_Config2__Quantity__c;
            objAssetWrapperClass.StartDate = aliSO.
Apttus_Config2__StartDate__c;
            objAssetWrapperClass.EndDate = aliSO.
Apttus_Config2__EndDate__c;
            lstAssetWrapper.add(objAssetWrapperClass);
        }
    }

    // if there are no assets by the search text that the user
entered,
    // display an appropriate message.
    else {
        lstAssetWrapper.clear();
```

```
            ApexPages.addMessage(new ApexPages.Message(ApexPages.
severity.info,
                                                    'No result found.
Please try again.'));
        }
}
```

Comparing Products

This is used to retrieve products, based on product IDs, and compare them side by side. This is typically done from the Product Catalog, to decide between products before adding them to the cart.

compareProducts

Parameters		
Name	**Type**	**Description**
request	CPQ.FeatureInfoRequestDO	The request data object.

Request Data Object - CPQ.FeatureInfoRequestDO		
Field	**Type**	**Description**
productIds	List<ID>	The list of Ids of the product you want to compare.

Response Data Object - CPQ.FeatureInfosResponseDO		
Field	**Type**	**Description**
FeatureInfos	List<Apttus_CPQApi.FeatureSupport.FeatureInfo>	This returns the list of features for each of the products included in the comparison.
ProductId	ID	ID of the products, which have a feature set to be compared.

Data Object - Apttus_CPQApi.FeatureSupport		
Field	**Type**	**Description**
ProductFeatureSOId	Id	Id of associated ProductFeature__c Salesforce Object
ProductFeatureSet	FeatureSet	Associated Product Feature Set

Data Object - Apttus_CPQApi. FeatureSupport. FeatureSet		
Field	**Type**	**Description**
FeatureSetSOId	Id	Id of associated Apttus_Config2__FeatureSet__cSalesforce Object
Name	String	Name of the associated Product Feature Set
Description	String	Description of associated product feature set
Features	List<ProductFeatureValue>	List of associated feature values
Sequence	Decimal	Order of the feature set

Data Object - Apttus_CPQApi. FeatureSupport. ProductFeatureValue		
Field	**Type**	**Description**
ProdFeatureValueSOId	Id	Product Feature Value SoId
ProdId	Id	ID of the product.
FeatureSOId	Id	ID of the associated feature set.

Data Object - Apttus_CPQApi. FeatureSupport. ProductFeatureValue		
Name	String	Name of the feature
Sequence	Decimal	The feature display sequence
Value	String	Product Feature Value
IsIncluded	Boolean	Whether it is included

Code Sample

The sample below enables you to fetch the IDs of the selected products. Two or more products should be selected to enable comparison. After the user selects two or more products such as a Laptop from different vendors, invoke this API.

```
public List<CPQ.FeatureInfoResponseDO> compareProducts(List<ID>
productIds)


{



                //Prompt
appropriate error message validation message when the productIds
parameter has
less than two products.


                if(productIds.size()
< 2)


                {


                                    System.debug('Please
select atleast two products to compare.');


                                    return
```

```
null;

                }


                //If
two or more products are selected execute the code below

                else

                {

                                Apttus_CPQApi.CPQ.
FeatureInfoRequestDO
request = new Apttus_CPQApi.CPQ.FeatureInfoRequestDO();

                                request.ProductIds
= productIds;




                                //Invoke
the compare products API

                                List<Apttus_CPQApi.CPQ.
FeatureInfoResponseDO>
response = Apttus_CPQApi.CPQWebService.compareProducts(request);

                                return
response;

                }

}
```

Code sample for creating a table in a Visualforce page to display the compared products:

```
'compareFeatureSets' in Visualforce code is the getter variable
which hold the
response from compareProducts apex method.


Visualforce code


<!-- Compare Products
Panel -->


<apex:outputPanel layout="block"
Id="compareProductsPanel"


styleClass="grid-2-13
clearfix compareProductsPanel">


<table>


<apex:repeat var="featureSetName"
value="{!compareFeatureSets}">


<tr>


<th data-priority="1"
colspan="{!colSpan}"><h4>{!featureSetName}<


/h4></th>


</tr>


<apex:repeat var="featureValue"
value="{!compareFeatureSets
```

```
[featureSetName][0].productFeatureSet.features}">


<apex:outputPanel layout="none"
rendered="{!hasIncludedProduct


[compareFeatureSets[featureSetName][0].productFeatureSet.


featureSetSOId]}">


<tr>


<th class="label">{!featureValue.Name}</th>


<apex:repeat var="product"
value="{!productIds}">


<apex:repeat var="prodFeatureInfo"
value="{!


productFeatureInfosResponse[product]}">


<apex:outputPanel layout="none"
rendered="{!


prodFeatureInfo.productFeatureSet.Name
== featureSetName}" >


<apex:repeat var="prodFeatureValue"
value="{!


prodFeatureInfo.productFeatureSet.features}">


<apex:outputPanel layout="none"
rendered="{!
```

```
prodFeatureValue.featureSOId==
featureValue.featureSOId &&



prodFeatureValue.isIncluded}"
>


<td style="border:
1px #CCCCCC solid;">{!prodFeatureValue.Value}</td>


</apex:outputPanel>


<apex:outputPanel layout="none"
rendered="{!


prodFeatureValue.featureSOId
== featureValue.featureSOId && NOT


(prodFeatureValue.isIncluded)}"
>


<td style="border:
1px #CCCCCC solid;">--


</td>


</apex:outputPanel>


</apex:repeat>


</apex:outputPanel>


</apex:repeat>
```

```
</apex:repeat>

</tr>

</apex:outputPanel>

</apex:repeat>

</apex:repeat>

</table>

</apex:outputPanel>
```

Adding a Bundle to a Cart

This adds a bundle along with selected products, options, and updated quantity to the cart.

addBundle

Parameters		
Name	**Type**	**Description**
request	CPQ.AddBundleRequestDO	The id of the agreement

Request Data Object - CPQ.AddBundleRequestDO		
Field	**Type**	**Description**
CartId	ID	The Id of the cart.

Request Data Object - CPQ.AddBundleRequestDO		
Field	**Type**	**Description**
SelectedBundle	CPQ.SelectedBundleDO	The bundle to be added to the cart.

Data Object - CPQ.SelectedBundleDO		
Field	**Type**	**Description**
SelectedProduct	CPQ.SelectedProductDO	The selected product data object.
SelectedOptions	List<CPQ. SelectedOptionDO>	The list of selected option data objects.

Data Object - CPQ.SelectedProductDO		
Field	**Type**	**Description**
ProductId	ID	Id of the product bundle.
Quantity	Decimal	The bundle quantity.
SellingTerm	Decimal	The bundle selling term.
StartDate	Date	The start date. You should ensure you use the correct date format.
EndDate	Date	The end date.
Comments	String	Comments associated with the record.
CustomFields	List<String> CustomFields	List of custom fields created for your product.
CustomData	Apttus_Config2__LineItem__c CustomData	This can be used to include the list of custom fields you have added to the product.

Data Object - CPQ.SelectedOptionDO		
Field	**Type**	**Description**
ComponentId	ID	Id of the component.
ComponentProductId	ID	Id of the component product.
Quantity	Decimal	The option quantity.
SellingTerm	Decimal	The option selling term.
StartDate	Date	The start date. You should ensure you use the correct date format.
EndDate	Date	The end date.
Comments	String	Comments associated with the object record.
CustomFields	List<String>	List of Custom Field's API Name
CustomData	Apttus_Config2__LineItem__c	The values to be set for the custom fields in the CustomFields List

Response Data Object - CPQ.AddBundleResponseDO		
Field	**Type**	**Description**
LineNumber	Decimal	The bundle line number.

Code Sample

The sample below enables you to add a bundled product with a specific product ID and its associated quantity to a specific cart with a specific cartID. For example, a user selects a bundled product-Laptop+Mouse and clicks Add to Cart, the request to add bundle to the cart is invoked and the bundle with the specific product ID is added to a cart with the cart ID.

```
/**
 * The below method demonstrates how to add a bundle product to an
existing cart (every quote has a cart)
 * Lets assume the Quote's Cart is blank and Laptop is a bundle
product and its two options are Keyboard and Mouse
 * The input of this method is Quote Number and the Id of the Laptop
bundle product
 * Inside the method we will add the Laptop bundle product and also
its two options Keyboard and Mouse to the cart
 */
public static void addBundle(String quoteNumber, ID bundleProductId)
{

    List<Apttus_Config2__ProductConfiguration__c> cart = [SELECT
Apttus_Config2__PriceListId__c,Id FROM
Apttus_Config2__ProductConfiguration__c WHERE
Apttus_QPConfig__Proposald__r.Name = :quoteNumber LIMIT 1];

    if(!cart.isEmpty() && bundleProductId != null) {

        // Assume the quantity and selling term for the bundle
product and its options is 1
        Integer quantity = 1;
        Integer sellingTerm = 1;

        // Create the request object
        Apttus_CPQApi.CPQ.AddBundleRequestDO request = new
Apttus_CPQApi.CPQ.AddBundleRequestDO();
        request.CartId = cart.get(0).Id;

        // Add the bundle product to the request
        request.SelectedBundle = new Apttus_CPQApi.CPQ.
SelectedBundleDO();
        request.SelectedBundle.SelectedProduct = new Apttus_CPQApi.
CPQ.SelectedProductDO();
        request.SelectedBundle.SelectedProduct.ProductId =
bundleProductId;
        request.SelectedBundle.SelectedProduct.Quantity = quantity;
        request.SelectedBundle.SelectedProduct.SellingTerm =
sellingTerm;
```

```
        // Get all the options of the bundle product
        Apttus_CPQApi.CPQ.ProductOptionGroupSearchResultDO
productOptionGroupResult = Apttus_CPQApi.CPQWebService.
getOptionGroupsForPriceListProduct(cart.get(0).
Apttus_Config2__PriceListId__c, bundleProductId);
        if(productOptionGroupResult.HasOptionGroups) {
            // Add the option products to the request
            request.SelectedBundle.SelectedOptions = new
List<Apttus_CPQApi.CPQ.SelectedOptionDO>();
            for(Apttus_CPQApi.CPQ.ProductOptionGroupDO
productOptionGroup : productOptionGroupResult.OptionGroups) {
                if(productOptionGroup.HasOptionComponents) {
                    for(Apttus_CPQApi.CPQ.ProductOptionComponentDO
productOptionComponent : productOptionGroup.OptionComponents) {
                        Apttus_CPQApi.CPQ.SelectedOptionDO
selectedOptionDO = new Apttus_CPQApi.CPQ.SelectedOptionDO();
                        selectedOptionDO.ComponentId =
productOptionComponent.ComponentId;
                        selectedOptionDO.ComponentProductId =
productOptionComponent.ComponentProductId;
                        selectedOptionDO.Quantity = quantity;
                        selectedOptionDO.SellingTerm = sellingTerm;
                        request.SelectedBundle.SelectedOptions.add
(selectedOptionDO);
                    }
                }
            }
        }

        // Execute the addBundle routine
        Apttus_CPQApi.CPQ.AddBundleResponseDO response =
Apttus_CPQApi.CPQWebService.addBundle(request);

        System.debug('Line Number of added bundle = ' + response.
LineNumber);
    }
}
```

Adding Options to a Bundle

This adds one or more Options Products to a Bundle product.

addoptions

Request Parameters		
Name	**Type**	**Description**
CartID	ID	The cart ID
lineNumber	Integer	Line Number of the Bundle Product
selectedOptions	List<Apttus_CPQApi.CPQ.SelectedOptionDO>	List of Options

Data Object - CPQ.SelectedOptionDO		
Field	**Type**	**Description**
ComponentId	ID	Id of the component.
ComponentProductId	ID	Id of the component product.
Quantity	Decimal	The option quantity.
SellingTerm	Decimal	The option selling term.
StartDate	Date	The start date. You should ensure you use the correct date format.
EndDate	Date	The end date.
Comments	String	

Data Object - CPQ.SelectedOptionDO		
Field	**Type**	**Description**
		Comments associated with the object record.
CustomFields	List<String>	List of Custom Field's API Name
CustomData	Apttus_Config2__LineItem__c	The values to be set for the custom fields in the CustomFields List

Code Sample

Using the sample below you can dynamically add options to a bundled product that the user selects in the cart line item. For example, if the user selects a bundled product, Laptop, in the cart line item you can fetch the configured options for that selected product using the addoptions API. In the sample below, for a selected product you initially fetch the productID. Using the retrieve option groups API, you can check whether the selected product has option groups associated. If an option group is associated, fetch the option components and create a list comprising the CPQ. SelectedOptionDO. Invoke the addoptions API and pass the cartID, productID, and the list CPQ. SelectedOptionDO.

```
 public void addOptions()
{
    string productName='';
    Integer lineNumber = 0;
    for(LineItemWrapperClass selProdWrap: lstWrapItems)
    {
//For a selected bundled product from a cart line item execute the
loop below
    if(selProdWrap.selected)
        {
            productName = selProdWrap.ProductName;
            lineNumber = Integer.valueOf(selProdWrap.LineNumber);
            break;
        }
    }
//If no product is selected, show an appropriate error message
    if(String.isBlank(productName))
```

```
    {
        ApexPages.addMessage(new ApexPages.Message(ApexPages.
severity.info, 'Please select at least one Product.'));
    }
    else
    {
        List<Apttus_CPQApi.CPQ.SelectedOptionDO> selectedOptDOList =
new List<Apttus_CPQApi.CPQ.SelectedOptionDO>();

//For the selected product fetch the product ID
        List<Product2> productId = [SELECT Id FROM Product2 WHERE
Name = :productName LIMIT 1];

//Execute the getoptionGroups API to fetch the options with option
groups
        Apttus_CPQApi.CPQ.ProductOptionGroupSearchResultDO result =
Apttus_CPQApi.CPQWebService.getOptionGroupsForPriceListProduct
(priceListId, productId[0].Id);

//If the selected product has option groups execute the loop
below
        if(result.HasOptionGroups)
        {
            List<Apttus_CPQApi.CPQ.ProductOptionGroupDO>
prodOptGrpDOList = result.OptionGroups;
            for(Apttus_CPQApi.CPQ.ProductOptionGroupDO prodOptGrpDO
: prodOptGrpDOList)
            {
//For an option group if a product has option components such as
quantity execute the loop below
                if(prodOptGrpDO.HasOptionComponents)
                {
                    List<Apttus_CPQApi.CPQ.ProductOptionComponentDO>
prodOptCompDOList = new List<Apttus_CPQApi.CPQ.
ProductOptionComponentDO>();
                    prodOptCompDOList = prodOptGrpDO.
OptionComponents;

//Fetch all the option components for a particular option group.
                    for(Apttus_CPQApi.CPQ.ProductOptionComponentDO
prodOptCompDO :prodOptCompDOList )
                    {
                        Apttus_CPQApi.CPQ.SelectedOptionDO
selectedOptDO = new Apttus_CPQApi.CPQ.SelectedOptionDO();
                        selectedOptDO.ComponentId = prodOptCompDO.
ComponentId;
```

```
                        selectedOptDO.ComponentProductId =
prodOptCompDO.ComponentProductId;
                        selectedOptDO.Quantity = 1;
                        selectedOptDOList.add(selectedOptDO);
                    }
                }
            }
        }
//Invoke the addOptions API and pass the cartID, selected product
line number, and the Option List retrieved from the getoptiongroups
API
        Apttus_CPQApi.CPQ.AddOptionsResponseDO addOptRespDO =
Apttus_CPQApi.CPQWebService.addOptions(cartId, lineNumber,
selectedOptDOList);

    }
}
```

Adding Products to a Cart

This API adds one or more products (with default options) to the cart along with quantity, term, start date, and end date.

addMultiProducts

Parameters		
Name	**Type**	**Description**
request	CPQ.AddMultiProduct RequestDO	The request data object.

Request Data Object - CPQ.AddMultiProductRequestDO		
Field	**Type**	**Description**
CartId	ID	The Id of the cart.
SelectedProducts	List <CPQ. SelectedProductDO>	The list of selected product data objects.

Response Data Object - CPQ.AddMultiProductResponseDO		
Field	**Type**	**Description**
LineNumbers	List<Decimal>	The list of line numbers added to the cart.

Data Object - CPQ.SelectedProductDO		
Field	**Type**	**Description**
ProductId	ID	Id of the product bundle.
Quantity	Decimal	The bundle quantity.
SellingTerm	Decimal	The bundle selling term.
StartDate	Date	The start date. You should ensure you use the correct date format.
EndDate	Date	The end date.
Comments	String	Comments associated with the record.
CustomFields	List<String> CustomFields	List of custom fields created for your product.
CustomData	Apttus_Config2__LineItem__c CustomData	This can be used to include the list of custom fields you have added to the product.

Code Sample

The sample below enables you to add multiple selected products with a specific product ID and its associated quantity, validity and selling term to a specific cart with a specific cartID. For example, if the user has selected software and hardware products, the products are added to the cart, on click of Add to Cart. The user is navigated to the cart page where they can view and change the quantity, selling term and other editable aspects of the product.

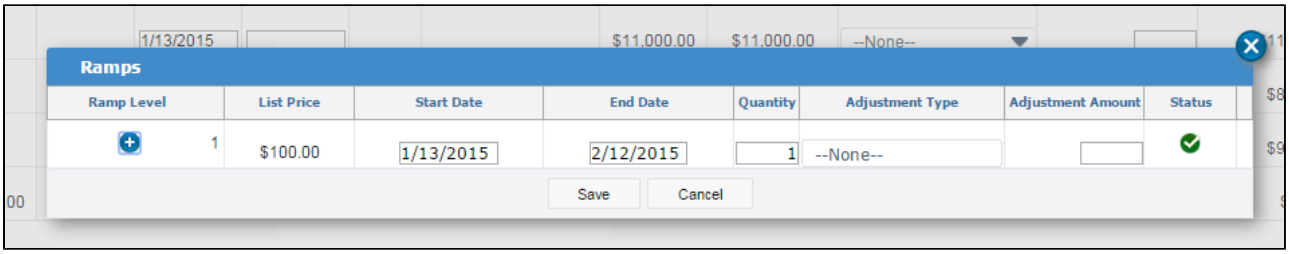
```
/**
 * The below method demonstrates how to add multiple products to an
existing cart (every quote has a cart)
 * Lets assume the Quote's Cart is blank and the standalone/bundle
products are Laptop, Monitor, Wifi Router
 * The input of this method is Quote Number and the Ids of the
Laptop bundle product, Monitor and Wifi Router standalone products
 */
public static void addMultipleProducts(String quoteNumber, List<ID>
productIds) {

    List<Apttus_Config2__ProductConfiguration__c> cart = [SELECT Id
FROM Apttus_Config2__ProductConfiguration__c WHERE
Apttus_QPConfig__Proposald__r.Name = :quoteNumber LIMIT 1];

    if(!cart.isEmpty() && productIds != null && !productIds.
isEmpty()) {

        // Assume the quantity and selling term for the product is 1
        Integer quantity = 1;
        Integer sellingTerm = 1;

        // Create the request object
        Apttus_CPQApi.CPQ.AddMultiProductRequestDO request = new
Apttus_CPQApi.CPQ.AddMultiProductRequestDO();
        request.CartId = cart.get(0).Id;

        // Add the products to the request
        for(ID productId : productIds) {
            Apttus_CPQApi.CPQ.SelectedProductDO selectedProduct = new
 Apttus_CPQApi.CPQ.SelectedProductDO();
            selectedProduct.ProductId = productId;
            selectedProduct.Quantity = quantity;
            selectedProduct.SellingTerm = sellingTerm;
            request.SelectedProducts.add(selectedProduct);
        }

        // Execute the addMultiProducts routine
        Apttus_CPQApi.CPQ.AddMultiProductResponseDO response =
Apttus_CPQApi.CPQWebService.addMultiProducts(request);

        System.debug('Line Numbers of added products = ' + response.
LineNumbers);
    }
}
```

Adding Price Ramps to a Cart (CPQ Web Service)

After you add a line item to a cart, this API enables you to add primary and secondary ramp line items for the line item. Once the ramp line items are created, you can also update the ramp line item details or delete ramp line item details using standard SOQL queries.

When you use Apttus CPQ out of the box, invoke the ramp using the red icon to the left of the primary line item.

Once you click the ramp icon, the ramp dialog appears:



The ramp dialog allows you to add, edit dates and quantity, make adjustments, save the changes, and cancel the changes.

Once the customer adds a ramp to a cart line item, they can do the following:

- Edit the start date, end date, quantity, adjustment type and adjustment amount based on the custom setting.
- The start date of a ramp line item defaults to the end date+1 of the previous line item.
- The end date of a ramp line item defaults to a date such that the difference between the start date and end date is the same as that of the previous line item.
- The user can add more ramp line items after or in between the ramp line items.
- The user can remove the new line before saving by clicking on the icon in the right most column.

Use the addMultiProducts API to add products to the cart.

addMultiProducts

Parameters		
Name	**Type**	**Description**
request	CPQ.AddMultiProduct RequestDO	The request data object.

Request Data Object - CPQ.AddMultiProductRequestDO		
Field	**Type**	**Description**
CartId	ID	The Id of the cart.
SelectedProducts	List <CPQ. SelectedProductDO>	The list of selected product data objects.

Response Data Object - CPQ.AddMultiProductResponseDO		
Field	**Type**	**Description**
LineNumbers	List<Decimal>	The list of line numbers added to the cart.

Data Object - CPQ.SelectedProductDO		
Field	**Type**	**Description**
ProductId	ID	Id of the product bundle.
Quantity	Decimal	The bundle quantity.
SellingTerm	Decimal	The bundle selling term.
StartDate	Date	The start date. You should ensure you use the correct date format.

Data Object - CPQ.SelectedProductDO		
Field	**Type**	**Description**
EndDate	Date	The end date.
Comments	String	Comments associated with the record.
CustomFields	List<String> CustomFields	List of custom fields created for your product.
CustomData	Apttus_Config2__LineItem__c CustomData	This can be used to include the list of custom fields you have added to the product.

Code Sample

The sample below enables you to add ramp line items after you have:

- Added Products to the cart using the AddMultiProducts APIs,
- Updated the Price for the added products using the updatePriceforCart API.
- Selected the products for which you want to add a ramp for.

Using the sample below you fetch the list of selected products for which you want to add a ramp. You also fetch the parameters for each of the selected products. For all the ramps you create, set PriceGroup as Price Ramp and PricingStatus as Pending. For a primary line item, set IsPrimaryLine__c = true, IsPrimaryRampLine__c = true, and PrimaryLineNumber__c = 1.

```
public void createRampLineItems()
    {
        List<String> rampLineItems = new List<String>();

        if(lstWrapItems.size() > 0)
        {
            // Create a list of selected products for which you want
to create a ramp for
            for(LineItemWrapperClass objLineItemWrapperClass :
lstWrapItems)
            {
                if(objLineItemWrapperClass.Selected)
```

```
                {
                    rampLineItems.add(objLineItemWrapperClass.Name);
                }
            }

            // Sort Ramp Line Items by name
            rampLineItems.sort();

            Integer rampLineItemIndex = 1;

            for(String rampLineItemName : rampLineItems)
            {
                // Get Line Item parameters for the selected products
                Apttus_Config2__LineItem__c lineItem = [SELECT
Apttus_Config2__ItemSequence__c, Apttus_Config2__PricingStatus__c,
Apttus_Config2__PriceGroup__c,
                Apttus_Config2__IsPrimaryRampLine__c,
Apttus_Config2__IsPrimaryLine__c, Apttus_Config2__LineNumber__c,
Apttus_Config2__PrimaryLineNumber__c from
Apttus_Config2__LineItem__c WHERE
                Name=:rampLineItemName];

                //Set the parameters for each of the line items
                lineItem.Apttus_Config2__PriceGroup__c = 'Price Ramp'
;
                lineItem.Apttus_Config2__PricingStatus__c = 'Pending'
;
                lineItem.Apttus_Config2__LineNumber__c = 1;
                lineItem.Apttus_Config2__PrimaryLineNumber__c = 1;
                lineItem.Apttus_Config2__ItemSequence__c =
rampLineItemIndex;

                //For a primary line item set the following
            `    if(rampLineItemIndex == 1)
                {
                    lineItem.Apttus_Config2__IsPrimaryLine__c = true;
                    lineItem.Apttus_Config2__IsPrimaryRampLine__c = t
rue;
                }

                //For all secondary line items set the following
parameters
                else
                {
                    lineItem.Apttus_Config2__IsPrimaryLine__c = false
;
```

```
                    lineItem.Apttus_Config2__IsPrimaryRampLine__c = f
alse;
                }

                // Update Line Items
                update lineItem;

                rampLineItemIndex++;
            }
        }
        else
        {
            ApexPages.addMessage(new ApexPages.Message(ApexPages.
severity.info, 'No line items available.'));
        }
    }
```

Associating Constraint Rules to a Cart

This associates constraints rules to a cart.

associateConstraintRules

Parameters		
Name	**Type**	**Description**
cartId	ID	The id of the cart.
existingPrimaryNumbers	List<Integer>	This is a collection of primary numbers for which rules are already attached. When existing primary numbers are supplied, rules are associated with the new lines; whereas when null or empty collection is supplied, rules are attached to all the line items.

Code Sample

The sample below enables you to associate a constraint rule to a product added as a line item in a cart. The sample below fetches the line numbers of the products in the cart and associates the constraint rule to the new line numbers of the cart.

If the list object for primary line items is null, then the constraint rules are associated with all the products in the line items of the cart.

For example if the delivery and packaging charge constraint rule is already applied to the existing products in the cart, you will provide a list of *collection of primary line items comprising selected products* to which the rules are already applied, thus enabling the delivery and packaging constraint rule to be associated with the new line items added to the cart.

```
public void associateConstraintRules()
{
   List<Apttus_Config2__LineItem__c> liSOList = [select
Apttus_Config2__ProductID__c,
   Apttus_Config2__LineNumber__c from Apttus_Config2__LineItem__c
where
   Apttus_Config2__ConfigurationId__c = :cartID];

   acrList = new List<Apttus_Config2__LineItem__c>();

   List<Integer> primaryLines = new List<Integer>();
   for(Apttus_Config2__LineItem__c liSO: liSOList)
   {
       for(Apttus_Config2__LineItem__c acrSO: acrList)
       {
            if(acrSO.Id == liSO.Id)
            {
                primaryLines.Add(liSO.Apttus_Config2__LineNumber__c.
intValue());
            }
       }
   }

   for(Apttus_Config2__LineItem__c liSO: liSOList)
   {
       for(LineItemWrapperClass objLineItemWrapperClass :
lstWrapItems)
       {
           if(objLineItemWrapperClass.Selected)
           {
               acrList.add(liSO);
               break;
           }
       }
   }
```

```
    Apttus_CPQApi.CPQWebService.associateConstraintRules(cartId,
primaryLines);
```

Applying Constraint Rules to a Cart

This processes and applies all the constraint rules.

Best Practice

It is recommended that you use the associateConstraintRules API before using this API. Directly using this API may not trigger the constraint rule.

applyConstraintRules

Parameters		
Name	**Type**	**Description**
cartId	ID	The id of the cart.
finalCheck	Boolean	If this is set to true, runs rules that are marked as check on finalization.

Code Sample

The sample below enables you to apply constraint rules to a cart with a specific cart ID. If you set the finalCheck flag as true, the constraint rules are run when you finalize the cart. If the flag is set as false, the rules are run before cart finalization. For example, if you want to add a Shipping Costs constraint rule only after the cart is finalized, set the finalCheck as true and the shipping costs constraint rule is applied once the user finalized the cart. If you want to apply constraint rules before the cart is finalized, set finalCheck as false. For example, if you set the finalCheck flag as false, installation charges are added along with a product selected before the cart is finalized.

```
public void applyConstraintRules()
{
    // For rules that are not marked as Check on Finalization
    Apttus_CPQApi.CPQWebService.applyConstraintRules(cartID, false);
```

```
    // For rules that are marked as Check on Finalization
    //Apttus_CPQApi.CPQWebService.applyConstraintRules(cartID, true);
}
```

Removing a Bundle from a Cart

This removes a product bundle and related line items, options from the cart.

removeBundle

Parameters		
Name	**Type**	**Description**
request	CPQ.RemoveBundleRequestDO	The request data object.

Request Data Object - CPQ.RemoveBundleRequestDO		
Field	**Type**	**Description**
CartId	ID	The Id of the cart.
LineNumber	Decimal	The bundle line number.

Response Data Object - CPQ.RemoveBundleResponseDO		
Field	**Type**	**Description**
IsSuccess	Boolean	Indicates the success of the operation.

Code Sample

The sample below enables you remove a selected bundle form the cart using the line number and a cart id. Invoke the API, when the user selects a bundle from the cart and click Remove or Delete. Based on the line number and the cartID the asset is removed from the cart. Update the new line items using the getLineItems function.

```
/**
 * The below method demonstrates how to remove a bundle product from
an existing cart (every quote has a cart)
 * Lets assume the Quote's Cart has a bundle product called Laptop
and its two options are Keyboard and Mouse
 * The input of this method is Quote Number and the line number of
the Laptop bundle product
 */
public static void removeBundle(String quoteNumber, Integer
lineNumber) {

    List<Apttus_Config2__ProductConfiguration__c> cart = [SELECT Id
FROM Apttus_Config2__ProductConfiguration__c WHERE
Apttus_QPConfig__Proposald__r.Name = :quoteNumber LIMIT 1];

    if(!cart.isEmpty() && lineNumber != null) {

        // Create the request object
        Apttus_CPQApi.CPQ.RemoveBundleRequestDO request = new
Apttus_CPQApi.CPQ.RemoveBundleRequestDO();
        request.CartId = cart.get(0).Id;
        request.LineNumber = lineNumber;

        // Execute the removeBundle routine
        Apttus_CPQApi.CPQ.RemoveBundleResponseDO response =
Apttus_CPQApi.CPQWebService.removeBundle(request);

        System.debug('Remove bundle from cart response status = ' +
response.IsSuccess);
    }
}
```

Removing Multiple Bundles from a Cart

This removes one or more products (with default options) or bundles from the cart along with quantity, term, start date, and end date.

removeMultiBundles

Parameters		
Name	**Type**	**Description**
request	CPQ.RemoveMultiBundlesRequestDO()	The request data object.

Request Data Object - CPQ.RemoveMultiBundlesRequestDO()		
Field	**Type**	**Description**
CartId	ID	The Id of the cart.
LineNumber	List<Decimal>	The List of LineNumbers that need to be removed from the cart.

Response Data Object - CPQ.RemoveMultiBundlesResponseDO		
Field	**Type**	**Description**
IsSuccess	Boolean	Indicates the success of the operation.

Code Sample

Use the sample below to remove one ore more products of the cart. When the user selects multiple products on the cart and clicks Delete or Remove, invoke this API. The multiple products are deleted from the cart. . Based on the line number and the cartID the selected assets are removed from the cart. Update the new line items using the getLineItems function.

```
/**
```

```
 * The below method demonstrates how to remove multiple products
from an existing cart (every quote has a cart)
 * Lets assume the Quote's Cart has a 3 products,
 * Laptop is a bundle product (line number 1) and Monitor and Wifi
Router are standalone products (line number 2 and 3 respectively)
 * The input of this method is Quote Number and the line numbers of
the Laptop bundle product and Monitor and Wifi Router are standalone
products
 */
public static void removeMultiBundles(String quoteNumber,
List<Integer> lineNumbers) {

    List<Apttus_Config2__ProductConfiguration__c> cart = [SELECT Id
FROM Apttus_Config2__ProductConfiguration__c WHERE
Apttus_QPConfig__Proposald__r.Name = :quoteNumber LIMIT 1];

    if(!cart.isEmpty() && lineNumbers != null && !lineNumbers.
isEmpty()) {

        // Create the request object
        Apttus_CPQApi.CPQ.RemoveMultiBundlesRequestDO request = new
Apttus_CPQApi.CPQ.RemoveMultiBundlesRequestDO();
        request.CartId = cart.get(0).Id;
        request.LineNumbers = lineNumbers;

        // Execute the removeMultiBundles routine
        Apttus_CPQApi.CPQ.RemoveMultiBundlesResponseDO response =
Apttus_CPQApi.CPQWebService.removeMultiBundles(request);

        System.debug('Remove bundle from cart response status = ' +
response.IsSuccess);
    }
}
```

Applying Constraint Rules to Deleted Products

This runs the rules that are related to the deleted products.

applyConstraintRulesOnDelete

Parameters		
Name	**Type**	**Description**
cartId	ID	The id of the cart.
deletedProductsIds	List<ID>	This is a list of product ids of the line item deleted from the cart.

Code Sample

The sample below enables you to delete any constraint rules applied to a product that is deleted by a customer. For example, for all products you have added an inclusion constraint rule which states whenever you add a product, an installation charge is included along with the product. When the user deletes the product from the cart, this API enables you to delete the associated installation constraint rule applied to the product. Using the Product_ID parameter you can specify the product from which the constraint rule is to be disassociated from.

```
Apttus_CPQApi.CPQWebService.applyConstraintRulesOnDelete(CART_ID, new
 String[] {PRODUCT_IDS});
```

Retrieving Constraint Rules Results

This retrieves the constraints rules applied to a cart.

getConstraintRuleResult

Parameters		
Name	**Type**	**Description**
CartId	ID	When you pass the cartId as an argument to the API, it fetches ConstraintResultDO, which comprises all the constraint rules applied to the cart.

Response Data Object - CPQ.ConstraintResultDO		
Field	**Type**	**Description**
CartId	ID	The Id of the cart.
NeedMoreProcessing	Boolean	Indicates whether the rule required more processing.
HasPendingError	Boolean	Indicates whether there are pending rule actions that are error type.
HasPendingWarning	Boolean	Indicates whether there are pending rule actions that are warning type.
ConstraintRuleActions	List<CPQ. AppliedActionDO>	The list of constraint rule action applied to the cart.

Code Sample

The sample below enables you to fetch the constraint rules applied to a cart with a specific Cart ID. Use this API when you want to view the constraint rules applied to a cart. For example, to generate a report about the constraint rules applied to a cart, you can fetch the constraint rules using this API.

```
public void getConstraintRuleResult()
{
    Integer numErrors = 0;

    Apttus_CPQApi.CPQ.ConstraintResultDO constraintResult =
Apttus_CPQApi.CPQWebService.getConstraintRuleResult(cartID);

    List<Apttus_CPQApi.CPQ.AppliedActionDO> appliedActionDOList =
constraintResult.ConstraintRuleActions;

    for(Apttus_CPQApi.CPQ.AppliedActionDO appliedActDO:
appliedActionDOList)
    {
```

```
        ApexPages.addMessage(new ApexPages.Message(ApexPages.
severity.info, appliedActDO.Message));
        if(appliedActDO.MessageType.equals('Error') && appliedActDO.
IsPending)
        {
            numErrors++;
        }
    }
}
```

Computing the Net Price for a Bundle

This API is used to calculate the price for the individual and summary line items of the cart. It runs the pricing rules, calculate bundle, line item, option level net prices and update the cart line with calculated prices. After the pricing is calculated for the cart line items, the resulting information is stored in the *Line Item* and *Summary Group* objects.

This API will also apply all the relevant rules and calculate the pricing including following among others:

- Line Item level discount
- Total Price based on specified quantity
- Applied Tiered Pricing
- Apply Ramp Pricing
- Apply Contractual Pricing
- Apply Related Pricing Rules

computeNetPriceForBundle

Parameters		
Name	**Type**	**Description**
request	CPQ. ComputeNetPriceRequestDO	The request data object.

Request Data Object - CPQ.ComputeNetPriceRequestDO		
Field	**Type**	**Description**
CartId	ID	The Id of the cart.
LineNumber	Decimal	The bundle line number.

Response Data Object - CPQ.ComputeNetPriceResponseDO		
Field	**Type**	**Description**
IsSuccess	Boolean	Indicates whether getting the total net price was successful.

Code Sample

The sample below prompts the user with the appropriate messages if a cart does not exist or when no products exist in the cart. If the user clicks calculate net price and products exist in the cart, the API computes the net price for all the line numbers of products in the cart with a specific cartID. For example, if the user wants to check the aggregate price of all the products in the cart, you can create a **Total Net Price** button on your cart page and invoke the computeNetPriceForBundle() API on click of the button. The API will calculate the total price taking into account any discount or pricing rules associated with the line items in the cart.

```
 public void computeNetPriceForBundle()
{
//If a cart does not exist, there is no cartID. Show the following
message to the user.
    if(cartId == null)
    {
        ApexPages.addMessage(new ApexPages.Message(ApexPages.
severity.info, 'Please create cart.'));
    }
//If no products are added to the cart, the lineNumber of the cart
is null. Prompt the use to add products to the cart.
    else if(lineNumber == null)
    {
```

```
        ApexPages.addMessage(new ApexPages.Message(ApexPages.
severity.info, 'Please add a bundle to the cart.'));
    }
    else
    {
//Selected products are added to the cart. The request object
comprises the id of the cart and the line number in the cart
        for(Decimal line: lineNumber)
        {
            Apttus_CPQApi.CPQ.ComputeNetPriceRequestDO request = new
Apttus_CPQApi.CPQ.ComputeNetPriceRequestDO();
            request.CartId = cartId;
            request.LineNumber = line;

            Apttus_CPQApi.CPQ.ComputeNetPriceResponseDO
priceResponse = Apttus_CPQApi.CPQWebService.computeNetPriceForBundle
(request);

            ApexPages.addMessage(new ApexPages.Message(ApexPages.
severity.info, 'Success: ' + priceResponse.IsSuccess));
        }
    }
}
```

Finalizing a Cart's Contents

This finalizes the cart, synchronizing the cart line items with the quote/proposal.

finalizeCart

Parameters		
Name	**Type**	**Description**
request	CPQ.FinalizeCartRequestDO	The request data object.

Request Data Object - CPQ.FinalizeCartRequestDO		
Field	**Type**	**Description**
CartId	ID	The Id of the cart.

Response Data Object - CPQ.FinalizeCartResponseDO		
Field	**Type**	**Description**
IsSuccess	Boolean	Indicates whether finalizing the cart items was successful.

Code Sample

The sample enables you to finalize a cart with a specific cartID and synchronizes the products (line items) added to the cart with the quote or proposal used to generate the cart. You can create a button finalize cart which the user will click once the products in the cart are adjusted and final. Invoke this API on click of the button.

```
 public void finalizeCart()
    {
        // create the finalize cart request
        Apttus_CpqApi.CPQ.FinalizeCartRequestDO request = new
Apttus_CpqApi.CPQ.FinalizeCartRequestDO();
        // add request parameters
        request.CartId = CartId;
        // finalize the cart
          Apttus_CpqApi.CPQ.FinalizeCartResponseDO response =
Apttus_CpqApi.CPQWebService.finalizeCart(request);
    }
```

Synchronizing a Cart

This is used to sync the shopping cart with the product configuration on the quote record.

SynchronizeCart

Parameters		
Name	**Type**	**Description**
request	CPQ.SynchronizeCartRequestDO	The request data object.

Request Data Object - CPQ.SynchronizeCartRequestDO		
Name	**Type**	**Description**
CartId	ID	The ID of the cart (Product Configuration) you want to synchronize.

Response Data Object - CPQ.SynchronizeCartResponseDO		
Name	**Type**	**Description**
IsSuccess	Boolean	Indicates whether synchronzing the cart items was successful.

Code Sample

The sample below enables you to synchronize the cart items with a quote. When the customer has finalized the cart, you can synchronize the products in the cart with the quote used to generate the cart. Invoke this API when the user has finalized the cart. The cartID with the updated products is provided as a parameter to the API.

```
 public void synchronizeCart()
{
//Pass the cartID as a parameter to the API to synchronize your
quote with the updated cart items.
    Apttus_CPQApi.CPQ.SynchronizeCartRequestDO request = new
Apttus_CPQApi.CPQ.SynchronizeCartRequestDO();
    request.CartId  = cartId;

    Apttus_CPQApi.CPQ.SynchronizeCartResponseDO response =
Apttus_CPQApi.CPQWebService.synchronizeCart(request);
```

```
    ApexPages.addMessage(new ApexPages.Message(ApexPages.severity.
info, 'Success: ' + response.IsSuccess));
}
```

Abandoning a Cart

This deletes the selected cart in **Draft** status.

abandonCart

Parameters		
Name	**Type**	**Description**
request	CPQ.AbandonCartRequestDO	The request data object.

Request Data Object - CPQ.AbandonCartRequestDO		
Field	**Type**	**Description**
CartId	ID	The Id of the cart.

Response Data Object - CPQ.AbandonCartResponseDO		
Field	**Type**	**Description**
IsSuccess	Boolean	Indicates whether the cart is deleted successfully.

Code Sample

The sample below enables you to send an API request for abandon cart along with the cart ID. The cart whose id has been passed will be abandoned when the request is successful. The response message in the sample indicates whether the request was successful. When the user clicks Abandon Cart the abandonCart API will be invoked and the cartID associated with the request is abandoned.

```
public void abandonCart()
{
    Apttus_CPQApi.CPQ.AbandonCartRequestDO request = new
Apttus_CPQApi.CPQ.AbandonCartRequestDO();
    request.CartId = cartId;

    Apttus_CPQApi.CPQ.AbandonCartResponseDO response = Apttus_CPQApi.
CPQWebService.abandonCart(request);
    ApexPages.addMessage(new ApexPages.Message(ApexPages.severity.
info, 'Success: ' + response.IsSuccess));
}
```

Updating Quote Terms

This method enables you to update the expected start date and expected end date of the quote, along with its selling term. This method does the following:

- Updates *Expected Start Date* and *Expected End Date* for the Apttus Quote/Proposal object.
- Retrieves the finalized cart associated with the quote.
- Creates a new cart by checking out the finalized cart.
- Updates Expected Start Date & Expected End Date in the cart object. Note this is for the new *Apttus Product Configuration* cart object.
- Updates Start Date, End Date, and Selling Term on line item terms that have a valid start date, i.e. when the start date is not null.
- Selling Term may be set to null to clear out the existing selling term which triggers a recalculation of the selling term using the start and end dates. Alternatively, if start date and selling term are provided, the end date is automatically calculated.
- Updates the net price and total price for the cart.
- The ID of the new cart is returned to the caller.
- It is the caller's responsibility to finalize the cart.

The field *QuoteLineItemColl*, which is part of the *CPQ.UpdateQuoteTermRequest*, can accept a list of quote line item sObjects which can hold the data to override one or more line items. For this the line items need to have the following fields populated:

- Start Date
- End Date
- Selling Term
- Derived Form

Line items which match the *Derived From* value will get overriden values from the Quote Line items. The remaining fields will default to values provided in the request object. A null value in the *Start Date* field in the request will prevent the values from defaulting to the line items from the request object.

updateQuoteTerm

Parameters		
Name	**Type**	**Description**
CPQ.UpdateQuoteTermRequestDO	request	This is the request call made by the method.

Request Data Object - CPQ.UpdateQuoteTermRequestDO			
Field	**Type**	**Mandatory**	**Description**
QuoteId	ID	Yes	The Id of the quote you want to change the dates and selling term for.
QuoteLineItemCollDO	List of LineItems	No	The collection of Line items to be updated.
StartDate	Date	Yes	The expected start date of the quote.
EndDate	Date	No for Auto-roll scenario	The expected end of the quote.

Request Data Object - CPQ.UpdateQuoteTermRequestDO			
Field	**Type**	**Mandatory**	**Description**
SellingTerm	Decimal	No. Null for pro-rata	The selling term of the quote is typically measured in years, months, or days.
Recalculate Selling Term	Boolean	No for Auto-roll scenario. Yes for Pro-rata scenario	Set this value to true in the prorata scenario. For example, license validity start date will be applicable from the day the customer accepts the proposal.

Response Data Object - CPQ.UpdateQuoteTermResponseDO		
Field	**Type**	**Description**
CartId	ID	Id of the cart updated.

Code Sample

The sample below enables you update the end date of a cart after the date of acceptance of the proposal. The auto-roll up end date is the sum of the proposal acceptance start date and selling term. When the recipient of the proposal accepts the quote and updates the status, invoke this API to update the start date and selling term of the products associated to the quote. In the sample below, the user enters the quote name in a search field and clicks Search. If a valid quote exists by the name, the quote is fetched. If the user clicks the update Quote Terms button, the API is invoked and the end date is updated.

```
public void autoroll()
{
//Search a quote by its name and fetch its ID
    List<Apttus_Proposal__Proposal__c> listQuote = [Select Id from
Apttus_Proposal__Proposal__c where Name=:quoteNumber LIMIT 1];

//If a valid quote exists with the name execute the if loop
      if(listQuote.size() > 0)
```

```
    {
        Apttus_CPQApi.CPQ.UpdateQuoteTermRequestDO updateQTRequest =
new Apttus_CPQApi.CPQ.UpdateQuoteTermRequestDO();
        updateQTRequest.QuoteId = listQuote[0].Id;

//Start date=current date entered in the from field of the proposal.
        updateQTRequest.StartDate = fromdate;
//End date=startdate+selling term
        updateQTRequest.EndDate = updateQTRequest.StartDate.addMonths
(sellingTerm);
        /**updateQTRequest.EndDate = endDate;**/

        /*updateQTRequest.SellingTerm = 12;*/

//Execute the API to update quote term
        Apttus_CPQApi.CPQ.UpdateQuoteTermResponseDO updateQTReponse
= Apttus_CPQApi.CPQWebService.updateQuoteTerm(updateQTRequest);

        ID cartID = updateQTReponse.cartID;

//Invoke the finalize cart request to update end date with the cart
        Apttus_CPQApi.CPQ.FinalizeCartRequestDO finalizeRequest = new
 Apttus_CPQApi.CPQ.FinalizeCartRequestDO();
        finalizeRequest.CartId = cartId;
//Invoke the finalize cart API to update the cart details
      Apttus_CPQApi.CPQ.FinalizeCartResponseDO finalizeResponse =
Apttus_CPQApi.CPQWebService.finalizeCart(finalizeRequest);
    }
}
```

The sample below enables you to update the start date and end date of a cart after the date of acceptance of the proposal. When the recipient of the proposal accepts the quote and updates the status, invoke this API to update the start date, end date and selling term of the products associated to the quote. In the sample below, the user enters the quote name in a search field and clicks Search. If a valid quote exists by the name, the quote is fetched. If the user clicks the update Quote Terms button, the API is invoked and the start date is updated. The validity i.e the selling term has to be updated on a pro-rata basis.

```
public void prorata()
{
//Search a quote by its name and fetch its ID
    List<Apttus_Proposal__Proposal__c> listQuote = [Select Id from
Apttus_Proposal__Proposal__c where Name=:quoteNumber LIMIT 1];
```

```
//If a valid quote exists with the name execute the if loop
    if(listQuote.size() > 0)
    {


        Apttus_Proposal__Proposal__c propSO = [select id,
Apttus_Proposal__ExpectedStartDate__c,
Apttus_Proposal__ExpectedEndDate__c from
Apttus_Proposal__Proposal__c where id = :listQuote[0].Id limit 1];
        Date startDate = propSO.
Apttus_Proposal__ExpectedStartDate__c;
        Date endDate = propSO.Apttus_Proposal__ExpectedEndDate__c;

        Apttus_CPQApi.CPQ.UpdateQuoteTermRequestDO updateQTRequest =
new Apttus_CPQApi.CPQ.UpdateQuoteTermRequestDO();
        updateQTRequest.QuoteId = listQuote[0].Id;
//Fetch the start date of the proposal
        updateQTRequest.StartDate = fromdate;
        /**updateQTRequest.EndDate = updateQTRequest.StartDate.
addMonths(12);**/
//Fetch the end date of the proposal
        updateQTRequest.EndDate = endDate ;
        updateQTRequest.SellingTerm = null;

//Selling Term to be recalculated based on the start date and end
date
        updateQTRequest.ReCalcSellingTerm = true;
        /*updateQTRequest.SellingTerm = 12;*/

//Execute the API to update quote term
        Apttus_CPQApi.CPQ.UpdateQuoteTermResponseDO updateQTReponse
= Apttus_CPQApi.CPQWebService.updateQuoteTerm(updateQTRequest);

        ID cartID = updateQTReponse.cartID;
//Invoke the finalize cart request to update selling term of product
with the cart
        Apttus_CPQApi.CPQ.FinalizeCartRequestDO finalizeRequest = new
 Apttus_CPQApi.CPQ.FinalizeCartRequestDO();
        finalizeRequest.CartId = cartId;
        Apttus_CPQApi.CPQ.FinalizeCartResponseDO finalizeResponse =
Apttus_CPQApi.CPQWebService.finalizeCart(finalizeRequest);
    }
}
```

Price Breakup for a Cart or Specific Line Item

This method can be used to retrieve the price breakup for a cart or specific line item.

The *CartId* is a mandatory parameter, while *LineItemId* can be null.

getPriceBreakup

Parameters		
Name	**Type**	**Description**
request	CPQ.GetPriceBreakupRequestDO	The request data object.

Request Data Object - CPQ.GetPriceBreakupRequestDO		
Name	**Type**	**Description**
CartId	ID	The ID of the new cart object. Mandatory parameter.
LineItemId	ID	The ID of the line item. This can be null. If it is null the price breakup for all applicable line items in the cart is retrieved. If a value is provided, only the price breakup for that specific line item is retrieved.

Response Data Object - CPQ.GetPriceBreakupResponseDO		
Name	**Type**	**Description**
HasPriceBreakups	Boolean	Indicates whether there are price breakups for the cart.
PriceBreakups	List <CPQ. PriceBreakupCollDO>	The list of price breakup collection objects

Data Object - CPQ.PriceBreakupCollDO		
Name	**Type**	**Description**
LineItemId	ID	The ID of the line item associated with the price break sobjects.
BreakupItems	List<Apttus_Config2__PriceBreakup__c>	The list of price breakup sobjects associated with the line item.

Code Sample

Using the sample below you can enable the end user to view the price break up for a selected product. Suppose the user has selected a product Laptop, using the Price Breakup API, you can show the end user the pricing breakup. For example, if the user has bought 25 laptops and for 20 quantities of the laptop a 5% discount is provided on the net price, and for the other 5 quantities a discount of 10% is given. Using the getPriceBrakup API you can show the user the tiered pricing pattern applied to a product.

```
public void getPriceBreakup()
    {
        lstPriceBreakup = new List<PriceBreakUpWrapperClass>();
//For a selected product in a cart execute this loop.
        for(LineItemWrapperClass objLineItemWrapperClass:
lstWrapItems)
        {
            if(objLineItemWrapperClass.selected)
            {
                Apttus_CPQApi.CPQ.GetPriceBreakupRequestDO
priceBreakUpRequest = new Apttus_CPQApi.CPQ.
GetPriceBreakupRequestDO();
                priceBreakUpRequest.CartId = cartId;
                priceBreakUpRequest.LineItemId =
objLineItemWrapperClass.Id;

                //Pass the parameters to the API
                Apttus_CPQApi.CPQ.GetPriceBreakupResponseDO
priceBreakUpResponse = Apttus_CPQApi.CPQWebService.getPriceBreakup
(priceBreakUpRequest);
```

```
//If the response returns true i.e. HasProducts=true execute the
loop below and fetch the priceBreakup list.
                if(priceBreakUpResponse.HasPriceBreakups)
                {
                    lstPriceBreakUpCalls = new List<Apttus_CPQApi.
CPQ.PriceBreakupCollDO>();
                    lstPriceBreakUpCalls = priceBreakUpResponse.
PriceBreakups;

                    for(Apttus_CPQApi.CPQ.PriceBreakupCollDO
objPriceBreakUpCall : lstPriceBreakUpCalls )
                    {
                        //priceBreakUpList = new
List<Apttus_Config2__PriceBreakup__c>();
                        //priceBreakUpList = objPriceBreakUpCall.
BreakupItems;

                        String lineItemID = objPriceBreakUpCall.
LineItemId;

                        List<Apttus_Config2__PriceBreakup__c>
liSOpriceBreakUpList = [select id, Apttus_Config2__Sequence__c,
Apttus_Config2__BreakupType__c,
                        Apttus_Config2__TierStartValue__c,
Apttus_Config2__TierEndValue__c, Apttus_Config2__TierQuantity__c,
Apttus_Config2__TierBasePrice__c,
Apttus_Config2__TierExtendedPrice__c from
Apttus_Config2__PriceBreakup__c where Apttus_Config2__LineItemId__c
= :lineItemID];

                        for(Apttus_Config2__PriceBreakup__c
lipriceBreakUpList : liSOpriceBreakUpList)
                        {
                            PriceBreakUpWrapperClass
objPriceBreakUpWrapperClass = new PriceBreakUpWrapperClass();
                            objPriceBreakUpWrapperClass.BreakUpID =
lipriceBreakUpList.Id;
                            objPriceBreakUpWrapperClass.Sequence =
lipriceBreakUpList.Apttus_Config2__Sequence__c;
                            objPriceBreakUpWrapperClass.BreakupType
= lipriceBreakUpList.Apttus_Config2__BreakupType__c;
                            objPriceBreakUpWrapperClass.
TierStartValue = lipriceBreakUpList.
Apttus_Config2__TierStartValue__c;
                            objPriceBreakUpWrapperClass.TierEndValue
= lipriceBreakUpList.Apttus_Config2__TierEndValue__c;
```

```
                            objPriceBreakUpWrapperClass.TierQty =
lipriceBreakUpList.Apttus_Config2__TierQuantity__c;
                            objPriceBreakUpWrapperClass.
TierUnitPrice = lipriceBreakUpList.Apttus_Config2__TierBasePrice__c;
                            objPriceBreakUpWrapperClass.
TierUnitPrice = objPriceBreakUpWrapperClass.TierUnitPrice.setScale(2,
System.RoundingMode.HALF_UP);
                            objPriceBreakUpWrapperClass.
TierExtendedPrice = lipriceBreakUpList.
Apttus_Config2__TierExtendedPrice__c;
                            objPriceBreakUpWrapperClass.
TierExtendedPrice = objPriceBreakUpWrapperClass.TierExtendedPrice.
setScale(2, System.RoundingMode.HALF_UP);

                            lstPriceBreakup.add
(objPriceBreakUpWrapperClass);
                        }

                    }
                }
//If the product selected has no price breakups show an appropriate
message.
                else
                {
                    ApexPages.addMessage(new ApexPages.Message
(ApexPages.severity.info, 'No Price Breakup Available.'));
                }
            }
        }
    }
```

Updating Price For A Cart

This method enables you to update the price for items in a given cart. Only line items in pending status are updated.

updatePriceForCart

Parameters		
Name	**Type**	**Description**
Request	CPQ.UpdatePriceRequestDO	This is the request data object.

Request Data Object - CPQ.UpdatePriceRequestDO		
Field	**Type**	**Description**
CartId	ID	The Id of the cart to update the price for.

Response Data Object - CPQ.UpdatePriceResponseDO		
Field	**Type**	**Description**
IsPricePending	Boolean	Indicates whether there are more items to update in future invocations. The caller needs to invoke the update price method in a separate transaction. This should be repeated until there are no more items pending.
PendingLineNumbers	List<Integer>	The list of line numbers in pending status after the method completes.
CompletedLineNumbers	List<Integer>	The list of line numbers representing line items which were updated in the current update price invocation.

Code Sample

The sample below enables you to update the net price in the cart for pending line items. In the sample below, with a single call of updatePriceForCart API, the API computes the net price for a single line item with status pending. updatePriceForCart API is invoked using <apex:actionPoller> command until compute net price operation is completed for all the line items in the cart. For example, there are 10 line items in the cart and you invoke updatePriceForCart API 10 times at regular interval of 5 seconds using <apex:actionPoller> command to complete the computing net price operation for all 10 line items.

```
 <apex:actionPoller action="{!doUpdatePrice}" rerender="MessageID"
interval="5" rendered="{!actionPollerActive}" enabled="{!
actionPollerActive}"/> .
```

In the sample below the action poller is run as soon as the page is loaded. The hasPendingitems flag returns false since no pending items have been added to the cart yet. If a valid cartID has been passed as an argument to the API and isPricePending returns true, run the action poller and invoke the updatePriceforCart API. The action poller does not run when hasPendingitems=false. You can invoke this API when the user adds discounts to the cart and clicks Reprice.

```
public void updatePriceforCart()
{
    hasPendingItems = false;
    actionPollerActive = true;

    if(String.isBlank(cartId))
    {
        ApexPages.addMessage(new ApexPages.Message(ApexPages.
severity.info, 'Please create cart.'));

        objUpdatePriceRequestDO = new Apttus_CpqApi.CPQ.
UpdatePriceRequestDO();
    }
    else
    {
        // create the update price request
        objUpdatePriceRequestDO = new Apttus_CpqApi.CPQ.
UpdatePriceRequestDO();

        // add request parameters
        objUpdatePriceRequestDO.CartId = cartId;

        // update price for the cart
        Apttus_CpqApi.CPQ.UpdatePriceResponseDO result =
Apttus_CpqApi.CPQWebService.updatePriceForCart
(objUpdatePriceRequestDO);

        hasPendingItems = result.IsPricePending;

        // Start the action poller. If isPricePending=true returns
true, run the action poller and the updatePrice API is invoked again
        if (hasPendingItems)
        {
            actionPollerActive = true;
            flagUpdatePrice = true;
        }
    }
}
```

```
// invoke the following in a new transaction using action poller
public void doUpdatePrice()
{

    // update price for the cart

    if(String.isNotBlank(cartId) && flagUpdatePrice == true)
    {
        // create the update price request
        objUpdatePriceRequestDO = new Apttus_CpqApi.CPQ.
UpdatePriceRequestDO();

        // add request parameters
        objUpdatePriceRequestDO.CartId = cartId;

        Apttus_CpqApi.CPQ.UpdatePriceResponseDO result =
Apttus_CpqApi.CPQWebService.updatePriceForCart
(objUpdatePriceRequestDO);

        hasPendingItems = result.IsPricePending;
        List<Integer> pendingLineNumbers = result.PendingLineNumbers;
        List<Integer> completedLineNumbers = result.
CompletedLineNumbers;

        if( hasPendingItems == false && completedLineNumbers.size() !
= 0 )
        {
            // stop the action poller
            actionPollerActive = false;
        }
        else if( hasPendingItems == true && pendingLineNumbers.size()
> 0 )
        {
            actionPollerActive = true;
        }
    }
}
```

Computing Shipping for Cart Line Items

This API enables you to calculate the shipping amount for an entire order and does not display the breakup for each line item.

computeShippingForCart

Parameters		
Name	**Type**	**Description**
request	Apttus_CPQApi.CPQ.ComputeShippingRequestDO	The compute shipping request.

Request Data Object - CPQ.ComputeShippingRequestDO		
Field	**Type**	**Description**
CartID	ID	The id of the cart.

Response Data Object - CPQ.ComputeShippingResponseDO		
Field	**Type**	**Description**
TotalShippingItem	Apttus_Config2__LineItem__c	The line item with total taxes.
ShippingResults	Map<ID, Apttus_Config2.CustomClass.ShippingResult>	The shipping result object populated by the line item id. This is only available when called from Apex.

Data Object - Apttus_Config2.CustomClass.ShippingResult		
Field	**Type**	**Description**
ShippingCharge	Decimal	

Data Object - Apttus_Config2.CustomClass.ShippingResult		
Field	**Type**	**Description**
		Shipping amount total of all the cart line items. This is the total shipping amount for the line item. Line item is identified by the sequence in which the shipping result is added. This needs to be calculated and populated by the implementation.
Handback	Object	Any Apex Object passed by the calling class and passed back to the calling class in the result. Handback is used only in case when the shipping charge needs to be calculated for an object other than Cart line item e.g. Billing Schedule line items. • For calculating shipping charges for a cart, there is no need to pass the Handback object. It can be blank • For calculating the shipping charges for objects other than line items, "Item" field will be blank and the actual object will be passed as handback object to identify the object for which shipping charges are calculated.

Code Sample

The sample below enables you to compute shipping for a cart with a valid cartID. Using the sample below, you can compute shipping for the entire cart using the values fetched from the shipping callback class. If the shipping charge is location dependent, ensure that the account associated with the order or proposal has a valid shipping to and billing to address.

For more information about Tax and Shipping scenarios refer, Updating Taxes and Shipping for an Order.

```
public void computeShippingForCart()
{
    if(String.isNotBlank(cartId))
    {
        // Start timer
        startTime = System.currentTimeMillis();
```

```
        // Create the request
        Apttus_CPQApi.CPQ.ComputeShippingRequestDO request = new
Apttus_CPQApi.CPQ.ComputeShippingRequestDO();

        // Add request parameters
        request.CartId = cartId;

        // Compute Shipping For Cart
        Apttus_CPQApi.CPQ.ComputeShippingResponseDO result =
Apttus_CPQApi.CPQWebService.computeShippingForCart(request);

        // End timer
        ResponseTime('computeShippingForCart');

        // Get Total Tax Item
        ApexPages.addMessage(new ApexPages.Message(ApexPages.
severity.info, 'Total Shipping Item : ' + result.TotalShippingItem));

        Map<ID, Apttus_Config2.CustomClass.ShippingResult>
shippingResults = new Map<ID, Apttus_Config2.CustomClass.
ShippingResult>();

        shippingResults = result.ShippingResults;

        for(Id sId : shippingResults.keySet()) {

            Apttus_Config2.CustomClass.ShippingResult shippingResult
= shippingResults.get(sId);

            ApexPages.addMessage(new ApexPages.Message(ApexPages.
severity.info, 'Shipping Charges : ' + shippingResult.
ShippingCharge));
        }

    } else {
        ApexPages.addMessage(new ApexPages.Message(ApexPages.
severity.info, 'Invalid or blank cart Id.'));
    }
}
```

Computing Taxes for Cart Line Items

This API enables you to calculate tax breakups for cart line item.

computeTaxForCart

Parameters		
Name	**Type**	**Description**
request	Apttus_CPQApi.CPQ.ComputeTaxRequestDO	The compute tax request object.

Request Data Object - CPQ.ComputeTaxRequestDO		
Field	**Type**	**Description**
CartID	ID	The id of the cart.

Response Data Object - CPQ.ComputeTaxResponseDO		
Field	**Type**	**Description**
TotalTaxItem	Apttus_Config2__LineItem__c	The line item with total taxes.
TaxResults	Map<ID, Apttus_Config2.CustomClass.TaxResult>	The Tax Result object populated by the line item id. This is only available when called from Apex.

Data Object - Apttus_Config2.CustomClass.TaxResult		
Field	**Type**	**Description**
TaxAmount	Decimal	This is the total amount for the line item. This is sum of the breakup tax amounts for the line item. Line item is identified by the sequence in which the tax result is added. This needs to be calculated and populated by the implementation.
TaxBreakups	List<Apttus_Config2__TaxBreakup>	

Data Object - Apttus_Config2.CustomClass.TaxResult		
Field	**Type**	**Description**
		Tax breakup provide the different components of the tax for a given line item. This needs to be implemented and provided by the implementation. Tax breakup has structure specified below.
Handback	Object	Any Apex Object passed by the calling class and passed back to the calling class in the result. Handback is used only in case when the tax needs to be calculated for an object other than Cart line item e.g. Billing Schedule line items. • For calculating tax for cart line items, there is no need to pass the Handback object. It can be blank • For calculating the tax for object other than line items, "Item" field will be blank and the actual object will be passed as handback object to identify object for which tax is calculated.

Data Object - Apttus_Config2__TaxBreakup		
Field	**Type**	**Description**
TaxType	String	Type of tax to be applied. For example, sales, custom, excise.
TaxRate	Decimal	Tax percentage to be applied

Data Object - Apttus_Config2__TaxBreakup		
Field	**Type**	**Description**
TaxAppliesTo	String	The field on which the tax is applied to. For example, net price.
TaxAmount	Decimal	The tax amount calculated based on the formulae defined in the callback.
BreakupType	String	If you specify the breakup type as Detail, then the quote or order line items show the detailed tax break up for a cart line item.

Tax Breakup Type Example

Breakup Type	Tax Type	Tax Rate	Tax Applies to	Tax Amount
Detail	State Tax	10	Net Price	100
Detail	City Tax	1	Net Price	10
Total				110

Code Sample

The sample below enables you to compute tax for a cart with a valid cartID. Using the sample below, you can compute tax for the entire cart using the values fetched from the tax callback class. The fields Taxable, Tax Inclusive and Tax Code are inherited from the quote or order line item. The tax code must be populated on the Price list item for each product and charge type combination. The Account and Account Location have a Tax Exempt indicator which overrides everything. For example, if the Account has a Tax Exempt flag enabled, then Tax is not calculated for the account. The system gives preference to location unless it is blank. If a product is not taxable or tax inclusive, the system will skip that line item.

For more information about Tax and Shipping scenarios refer, Updating Taxes and Shipping for an Order.

```
public void computeTaxForCart()
{
```

```
    if(String.isNotBlank(cartId))
    {


        // Create the request
        Apttus_CPQApi.CPQ.ComputeTaxRequestDO request = new
Apttus_CPQApi.CPQ.ComputeTaxRequestDO();

        // Add request parameters
        request.CartId = cartId;

        // Compute Tax for cart
        Apttus_CPQApi.CPQ.ComputeTaxResponseDO result =
Apttus_CPQApi.CPQWebService.computeTaxForCart(request);



        // Get Total Tax Item
        ApexPages.addMessage(new ApexPages.Message(ApexPages.
severity.info, 'Total Tax Item : ' + result.TotalTaxItem));

        Map<ID, Apttus_Config2.CustomClass.TaxResult> taxResults = ne
w Map<ID, Apttus_Config2.CustomClass.TaxResult>();
        lstTaxBreakUpWrapper = new List<TaxBreakUpWrapper>();

        taxResults = result.TaxResults;

        for(ID tId : taxResults.keySet()) {
            Apttus_Config2.CustomClass.TaxResult taxResult =
taxResults.get(tId);

            // Tax Amount
            ApexPages.addMessage(new ApexPages.Message(ApexPages.
severity.info, 'Tax Amount : ' + taxResult.TaxAmount));

            // Tax BreakUps
            for(Apttus_Config2__TaxBreakup__c taxBreakUp : taxResult.
TaxBreakups) {

                TaxBreakUpWrapper temp = new TaxBreakUpWrapper();
                temp.TaxBreakUpId = taxBreakUp.Id;
                temp.TaxType = taxBreakUp.Apttus_Config2__TaxType__c;
                temp.TaxAppliedTo = taxBreakUp.
Apttus_Config2__TaxAppliesTo__c;
                temp.TaxRate = taxBreakUp.Apttus_Config2__TaxRate__c;
                temp.TaxAmount = taxBreakUp.
Apttus_Config2__TaxAmount__c;
```

```
                    temp.Sequence = taxBreakUp.
Apttus_Config2__Sequence__c;

                    lstTaxBreakUpWrapper.add(temp);
                }
            }

        } else {
            ApexPages.addMessage(new ApexPages.Message(ApexPages.
severity.info, 'Invalid or blank cart Id.'));
        }
}
```

Retrieving Incentives on the Cart

This API is used to retrieve the incentives which your users want to apply automatically or manually to the products in your shopping cart.

getIncentivesForCart

Parameters		
Name	**Type**	**Description**
request	CPQ.GetIncentivesForCartRequestDO	The request data object.

Request Data Object - CPQ.GetIncentivesForCartRequestDO		
Field	**Type**	**Description**
CartID	ID	The id of the cart, containing products on which the promotions are applied.

Response Data Object - CPQ.GetIncentivesForCartResponseDO		
Field	**Type**	**Description**
Incentives	List<Incentives>	This is a list of incentives, on which the PUSH promotion is applied.

Data Object - CPQ.GetIncentivesForCartResponseDO		
Field	**Type**	**Description**
Name	String	User specific name for the Incentive. This promotion name is displayed to the user at any place the Incentive information is displayed. For example, whenever a user looks-up the promotion information applicable to a specific product, the promotion name you provide here is displayed.
UseType	Picklist	This picklist indicates whether the incentive is of type Promotion or Rebate. In our case, select *Promotion*.
StopProcessingMoreIncentives	Boolean	Indicates if the system does not have to process other incentives in case if there are multiple incentives applicable at the same time.
Status	Picklist	Specifies the status of the Promotion. The status of the promotion is dependent on the promotion lifecycle. Once a promotion is approved, you can set the status of the approval as active.

Data Object - CPQ.GetIncentivesForCartResponseDO		
Field	**Type**	**Description**
		Approvals on Promotions - Admins can setup approval processes and other workflows on promotions. Marketing managers can send promotions for approval and approvers can view the promotion, approve or reject the promotions with or without comments, and activate it.
Sequence	Number	Defines the sequence in which incentives are applied, when there are more than one incentives.
RegionScope	Look up	Enables you to define promotions for a particular region. While defining a promotion you can choose to include or exclude a region. The region picklist must be defined for the Account object.
RegionScopeOper	Picklist	Consists of options like, Include and Exclude. These allow you to control if you want to include or exclude a region for a particular promotion.
ProductScope	Look up	Consists of the products for which you have defined promotions.
ProductScopeOper	Picklist	Based on the selected Product, you can control whether to include or exclude that product.
ProductFamilyScope	Look up	Consists of the product family for which you have defined promotions.
ProductFamilyScopeOper	Picklist	Based on the selected Product Family, you can control whether to include or exclude that particular product family.

Data Object - CPQ.GetIncentivesForCartResponseDO		
Field	**Type**	**Description**
PriceListScope	Look up	Consists of the price list for which you have defined promotions.
PriceListScopeOper	Picklist	Based on the selected Price List, you can control whether to include or exclude that particular price list and its products.
IncentiveNumber	Auto Number	System generated unique number for the Incentive. This differs from the Salesforce record id. When an active incentive is changed over time, the Incentive number remains same.
IncentiveCode	String	This is the code that is used for applying incentive. This code is captured on the order or proposal line items when an incentive is applied to the line item. By default, this value is set to Incentive Number, but is editable. This can also be used by the marketing team for marketing campaigns. When a promotion is not auto- applied, that is when the **Auto Apply ?** check box is cleared, the user can use the incentive code to avail a promotional offer.
ExpirationDate	Date	Date on which the promotion expires. It can be blank . It needs to be greater than the Start Date.
EffectiveDate	Date	Start Date from which the promotion is applicable.
CountryScope	Look up	Enables you to define promotions for a particular country. While defining a promotion you can choose to include or exclude a country. The country picklist must be defined for the Account object.
CountryScopeOper	Picklist	

Data Object - CPQ.GetIncentivesForCartResponseDO		
Field	**Type**	**Description**
		Consists of options like, Include and Exclude. These allow you to control if you want to include or exclude a country for a particular promotion.
AutoApply	Boolean	Can be set to *Yes* or *No*. Set it to *Yes* when you want a promotion to be automatically applied when a promotion criteria is met. Set it to *No* when you want the user to manually enter an Incentive Code to apply promotions.
ApplicationMethod	Picklist	When you select the Incentive category as *Promotion*, the incentive types are *Buy X Get X*, *Buy X Get Y*, *For Every X Get X*, *For Every Y Get Y*.
Active	Boolean	Can be set to *Yes* or *No*. Set it to *Yes* when you want a promotion to be applied and active for the criteria you specify. **Active** flag on the promotions will be set using a workflow linked to approvals similar to what you would do for any other custom object approval. For example, you can write a workflow rule to set the active status to true once the promotion is Approved (if approval is setup). Approvals set up would be similar to custom object approvals for any object.
AccountTypeScope	Look up	Enables you to define promotions for a particular Account type. While defining a promotion, you can choose to include or exclude the Account Type based on which the promotion is applied.
AccountTypeScopeOper	Picklist	Consists of options like, Include and Exclude. These allow you to define a promotion by choosing to include or exclude an account type.

Data Object - CPQ.GetIncentivesForCartResponseDO		
Field	**Type**	**Description**
AccountScope	Look up	Enables you to define promotions for a particular Account. While defining a promotion, you can choose to include or exclude the Account based on which the promotion is applied.
AccountScopeOper	Picklist	Consists of options like, Include and Exclude. These allow you to define a promotion by choosing to include or exclude an account.

Code Sample

After you configure your products and update your pricing, the following code allows you to fetch the promo code, which your users can apply on the products in your shopping cart. This code snippet is for the PUSH and PULL type promotions. In the PUSH type, the promo code is automatically applied and in the PULL type, the user enters the promo code for multiple products or on the entire cart before finalizing the cart. The Incentives list collects all the promo codes applicable on your shopping cart. The condition *Auto Apply__c == false* further filters only those promo codes which you want the user to apply manually on the products. Each incentive record captures all the details listed in the above table.

```
public void getIncentivesForCart()
    {
        Map<Decimal, String> incSequencePromoCodeMap = new
Map<Decimal, String>();
        List<Decimal> seqList = new List<Decimal>();

        Apttus_CPQApi.CPQ.GetIncentivesForCartRequestDO request = new
 Apttus_CPQApi.CPQ.GetIncentivesForCartRequestDO();
        system.debug('Cart ID: ' + cartId);
        request.CartID = cartId;

        Apttus_CPQApi.CPQ.GetIncentivesForCartResponseDO response =
Apttus_CPQApi.CPQWebservice.getIncentivesForCart(request);

        String couponCodes = '';
```

```
        if(response.Incentives.size() > 0)
        {
            for(Apttus_Config2__Incentive__c incSO : response.
Incentives)
            {
                if(incSO.Apttus_Config2__AutoApply__c == false)
                    couponCodes += incSO.
Apttus_Config2__IncentiveCode__c + ',';
            }
        }
    }
```

Cloning Bundle Line Items on the Cart

This method allows you to clone the primary bundle line items along with option line items and child line items (if applicable) on the Cart. This API is invoked when you click **Clone** action icon next to a bundle product on the Cart.

`cloneBundleLineItems`

Parameters		
Name	**Type**	**Description**
Request	CPQ.CloneLineItemsRequestDO	This is the request data object.

Request Data Object – CPQ.CloneLineItemsRequestDO		
Field	**Type**	**Description**
Cart ID	ID	The Id of the cart.
PrimaryLineNumber	Integer	The line numbers of the primary line items which have to be cloned using this API.

Response Data Object – CPQ.CloneLineItemsResponseDO		
Field	**Type**	**Description**

Response Data Object – CPQ.CloneLineItemsResponseDO		
PrimaryLineNumber	List<IntegerMapDO>	The list of primary line item numbers.

Response Data Object – CPQ.IntegerMapDO		
Field	**Type**	**Description**
mapItem.Key	Integer	The line item numbers of the source primary line item from which new line items are cloned using this API.
mapItem.Value	Integer	The line item numbers of the cloned line items.

Code Sample

Using the below sample, you can clone the primary bundle line items after you configure your products, add the attributes, configure your options and arrive on the cart. This API accepts the Cart ID and the primary bundle and option line item numbers. In the form of response, this API provides the original line items in the `mapItem.Key` parameter and the newly cloned line items in the `mapItem.Value` parameter.

For example, once you configure your cart by adding products using **`addBundle`** or **addMultiProducts** API, you can invoke this API in order to clone the bundle and option line items on the cart at any of the following:

- After the product is added on the Cart.
- Before making changes to pricing of the products.
- After updating the cart pricing using `updatePriceForCart` API.

```
/**
 * The below method demonstrates how to clone bundle product in an
existing cart (every quote has a cart)
 * Lets assume the Quote's Cart has a 3 products,
 * Laptop is a bundle product (line number 1) and Monitor and Wifi
Router are standalone products (line number 2 and 3 respectively)
 * The input of this method is Quote Number and the line number of
the Laptop bundle product
```

```
 */
public static void cloneBundleLineItems(String quoteNumber,
List<Integer> primaryLineNumbers) {

    List<Apttus_Config2__ProductConfiguration__c> cart = [SELECT Id
FROM Apttus_Config2__ProductConfiguration__c WHERE
Apttus_QPConfig__Proposald__r.Name = :quoteNumber LIMIT 1];

    if(!cart.isEmpty() && primaryLineNumbers != null && !
primaryLineNumbers.isEmpty()) {

        // Create the request object
        Apttus_CPQApi.CPQ.CloneLineItemsRequestDO request = new
Apttus_CPQApi.CPQ.CloneLineItemsRequestDO();
        request.CartId = cart.get(0).Id;
        request.PrimaryLineNumbers = primaryLineNumbers;

        // Execute the cloneBundleLineItems routine
        Apttus_CPQApi.CPQ.CloneLineItemsResponseDO response =
Apttus_CPQApi.CPQWebService.cloneBundleLineItems(request);

        for(Apttus_CPQApi.CPQ.IntegerMapDO intMapDO : response.
OriginalToCloneMap) {
            System.debug('Source Bundle Line Number = ' + intMapDO.
Key + ', Cloned Bundle Line Number = ' + intMapDO.Value);
        }
    }
}
```

Cloning Line Items on the Cart

This method enables you to clone the primary line items on the Cart. This API is invoked when you click **Clone** action icon next to a standalone product on the Cart.

`cloneLineItems`

Parameters		
Name	**Type**	**Description**
Request	CPQ.CloneLineItemsRequestDO	This is the request data object.

Request Data Object – CPQ.CloneLineItemsRequestDO		
Field	**Type**	**Description**
Cart ID	ID	The Id of the cart.
PrimaryLineNumber	Integer	The line item numbers of the primary line items which have to be cloned using this API.

Response Data Object – CPQ.CloneLineItemsResponseDO		
Field	**Type**	**Description**
PrimaryLineNumber	List<IntegerMapDO>	The list of primary line item numbers.

Response Data Object – CPQ.IntegerMapDO		
Field	**Type**	**Description**
mapItem.Key	Integer	The line item numbers of the source primary line item from which new line items are cloned using this API.
mapItem.Value	Integer	The line item numbers of the cloned line items.

Code Sample

Using the following sample, you can clone the primary line items after you configure your products, add the attributes and arrive on the cart. This API accepts the Cart ID and the primary line item numbers, and clones these line items. In the form of the response, this API provides the original line items in the `mapItem.Key` parameter and the newly cloned line items in the `mapItem.Value` parameter.

For example, once you configure your cart using *addMultiProducts* API, you can invoke this API in order to clone the line items on the cart at any of the following:

- After the product is added on the Cart.
- Before making changes to pricing of the products.
- After updating the cart pricing using *updatePriceForCart* API.

.

```
/**
 * The below method demonstrates how to clone standalone product in
an existing cart (every quote has a cart)
 * Lets assume the Quote's Cart has a 3 products,
 * Laptop is a bundle product (line number 1) and Monitor and Wifi
Router are standalone products (line number 2 and 3 respectively)
 * The input of this method is Quote Number and the line numbers of
the Monitor and Wifi Router are standalone products
 */
public static void cloneLineItems(String quoteNumber, List<Integer>
primaryLineNumbers) {

    List<Apttus_Config2__ProductConfiguration__c> cart = [SELECT Id
FROM Apttus_Config2__ProductConfiguration__c WHERE
Apttus_QPConfig__Proposald__r.Name = :quoteNumber LIMIT 1];

    if(!cart.isEmpty() && primaryLineNumbers != null && !
primaryLineNumbers.isEmpty()) {

        // Create the request object
        Apttus_CPQApi.CPQ.CloneLineItemsRequestDO request = new
Apttus_CPQApi.CPQ.CloneLineItemsRequestDO();
        request.CartId = cart.get(0).Id;
        request.PrimaryLineNumbers = primaryLineNumbers;

        // Execute the cloneLineItems routine
        Apttus_CPQApi.CPQ.CloneLineItemsResponseDO response =
Apttus_CPQApi.CPQWebService.cloneLineItems(request);

        for(Apttus_CPQApi.CPQ.IntegerMapDO intMapDO : response.
OriginalToCloneMap) {
            System.debug('Source Bundle Line Number = ' + intMapDO.
Key + ', Cloned Bundle Line Number = ' + intMapDO.Value);
        }
    }
}
```

Cloning Option Line Items on the Cart

This method allows you to clone the option line items and child line items (if applicable) on the Cart. This API is invoked when you click **Clone** action icon next to an option product on the Cart.

cloneOptionLineItems

Parameters		
Name	**Type**	**Description**
Request	CPQ.CloneLineItemsRequestDO	This is the request data object.

Request Data Object – CPQ.CloneLineItemsRequestDO		
Field	**Type**	**Description**
Cart ID	ID	The Id of the cart.
PrimaryLineNumber	Integer	The line numbers of the primary line items which have to be cloned using this API.

Response Data Object – CPQ.CloneLineItemsResponseDO		
Field	**Type**	**Description**
PrimaryLineNumber	List<IntegerMapDO>	The list of primary line item numbers.

Response Data Object – CPQ.IntegerMapDO		
Field	**Type**	**Description**
mapItem. Key	Integer	The line item numbers of the source primary line item from which new line items are cloned using this API.
	Integer	The line item numbers of the cloned line items.

Response Data Object – CPQ.IntegerMapDO		
mapItem. Value		

Code Sample

Using the below code sample, you can clone the option line items after you configure your bundle products, add the attributes and arrive on the cart. This API accepts the Cart ID and the option line item numbers, and clones these line items. In the form of the response, this API provides the original line items in the `mapItem.Key` parameter and the newly cloned line items in the `mapItem.Value` parameter.

For example, once you configure your cart by adding option products using *addoptions* API, you can invoke this API in order to clone the option line items on the cart at any of the following:

- After the product is added on the Cart.
- Before making changes to pricing of the products.
- After updating the cart pricing using *updatePriceForCart* API.

```
/**
 * The below method demonstrates how to clone option line items in
an existing cart (every quote has a cart)
 * Lets assume the Quote's Cart has a 3 products,
 * Laptop is a bundle product (line number 1) and Monitor and Wifi
Router are standalone products (line number 2 and 3 respectively)
 * The input of this method is Quote Number and the line number of
the Laptop bundle product
 */
public static void cloneOptionLineItems(String quoteNumber,
List<Integer> primaryLineNumbers) {

    List<Apttus_Config2__ProductConfiguration__c> cart = [SELECT Id
FROM Apttus_Config2__ProductConfiguration__c WHERE
Apttus_QPConfig__Proposald__r.Name = :quoteNumber LIMIT 1];

    if(!cart.isEmpty() && primaryLineNumbers != null && !
primaryLineNumbers.isEmpty()) {

        // Create the request object
        Apttus_CPQApi.CPQ.CloneLineItemsRequestDO request = new
Apttus_CPQApi.CPQ.CloneLineItemsRequestDO();
```

```
        request.CartId = cart.get(0).Id;
        request.PrimaryLineNumbers = primaryLineNumbers;

        // Execute the cloneOptionLineItems routine
        Apttus_CPQApi.CPQ.CloneLineItemsResponseDO response =
Apttus_CPQApi.CPQWebService.cloneOptionLineItems(request);

        for(Apttus_CPQApi.CPQ.IntegerMapDO intMapDO : response.
OriginalToCloneMap) {
            System.debug('Original Primary Line Number = ' +
intMapDO.Key + ', Cloned Primary Line Number = ' + intMapDO.Value);
        }
    }
}
```

Removing Line Items from the Cart

This API enables you to remove specific line items from the Cart. This API is invoked when you click **Remove** action icon next to any product on the Cart.

`removeLineItems`

Parameters		
Name	**Type**	**Description**
Request	CPQ.RemoveLineItemsRequestDO	This is the request data object.

Request Data Object – CPQ.RemoveLineItemsRequestDO		
Field	**Type**	**Description**
Cart ID	ID	The Id of the cart.
PrimaryLineNumbers	Integer	The line item numbers of the primary line items which have to be removed using this API.

Response Data Object – CPQ. RemoveLineItemsResponseDO		
Field	**Type**	**Description**
IsSuccess	Boolean	Indicates whether the operation of removing the line items was a success. True indicates success, false indicates that the line items were not removed from the Cart.

Code sample

You can use this API after you realize that the end user has added some products (bundle, standalone or option) and you want to remove those products from the Cart at once. Along with the primary line items, this API also removes the child line items under these primary line items (if applicable).

```
/**
 * The below method demonstrates how to remove multiple line items
from an existing cart (every quote has a cart)
 * Lets assume the Quote's Cart has 3 products,
 * a bundle product called Laptop (line number 1) and its two
options are Keyboard and Mouse
 * and Monitor and Wifi Router are standalone products (line number
2 and 3 respectively)
 * The input of this method is Quote Number and the line numbers of
the Laptop bundle product and Monitor standalone product
 */
public static void removeLineItems(String quoteNumber, List<Integer>
primaryLineNumbers) {

    List<Apttus_Config2__ProductConfiguration__c> cart = [SELECT Id
FROM Apttus_Config2__ProductConfiguration__c WHERE
Apttus_QPConfig__Proposald__r.Name = :quoteNumber LIMIT 1];

    if(!cart.isEmpty() && primaryLineNumbers != null && !
primaryLineNumbers.isEmpty()) {

        // Create the request object
        Apttus_CPQApi.CPQ.RemoveLineItemsRequestDO request = new
Apttus_CPQApi.CPQ.RemoveLineItemsRequestDO();
        request.CartId = cart.get(0).Id;
        request.PrimaryLineNumbers = primaryLineNumbers;
```

```
        // Execute the removeLineItems routine
        Apttus_CPQApi.CPQ.RemoveLineItemsResponseDO response =
Apttus_CPQApi.CPQWebService.removeLineItems(request);

        System.debug('Remove line items from cart response status = '
 + response.IsSuccess);
    }
}
```

Updating Category Hierarchy for a single Category

This method allows you to run the Category maintenance for a single category. This is applicable to only those categories which have products associated to them. You can invoke this API when a user clicks **Update View** button in the Category Hierarchy.

`updateCategoryView`

Parameters		
Name	**Type**	**Description**
hierarchyId	ID	This is the ID of the category hierarchy record.

Response Data Object – Apttus_CpqApi.BatchUpdateService		
Field	**Type**	**Description**
apexBatchJobId	ID	The ID of the Batch Job run.

Code sample

The code described below is used to run the Category Maintenance after you associate new products or modify the existing products under a category. You can also use this API when associate a Price List Item to a Category, under the related list, Price List Category.

This API takes up the ID of the category hierarchy record and returns the Batch job ID after success.

```
public void updateCategoryView(String categoryName)
{
```

```
    List<Apttus_Config2__ClassificationHierarchy__c> categoryList =
[SELECT ID FROM Apttus_Config2__ClassificationHierarchy__c WHERE
Name =: categoryName];
    String hierarchyId = categoryList[0].ID;

    ID apexBatchJobId = Apttus_CpqApi.BatchUpdateService.
updateCategoryView(hierarchyId);

    ApexPages.addMessage(new ApexPages.Message(ApexPages.severity.
info, 'Apex Batch Job Id : ' + apexBatchJobId));

}
```

Updating Category Hierarchy for multiple Categories

This method allows you to run the Category maintenance for multiple categories. This is applicable to only those categories which have products associated to them. You can invoke this API when a user clicks **Update View** button in the Category Hierarchy.

`updateCategoryViews`

Parameters		
Name	**Type**	**Description**
hierarchyIds	Set<ID>	This is the set containing the IDs of the category hierarchy records.

Response Data Object – Apttus_CpqApi.BatchUpdateService		
Field	**Type**	**Description**
apexBatchJobId	ID	The ID of the Batch Job run.

Code sample

The code described below is used to run the Category Maintenance for multiple Category Hierarchies after you associate new products or modify the existing products under multiple categories. You can also use this API when associate a Price List Item to a Category, under the related list, Price List Category.

This API takes up the set of IDs of the category hierarchy records and returns the Batch job ID after success.

```
public void updateCategoryViews(List<String> categoryNames)
{
    Set<ID> hierarchyIds = new Set<ID>();
    for(String categoryName : categoryNames)
    {
        List<Apttus_Config2__ClassificationHierarchy__c>
categoryList = [SELECT ID FROM
Apttus_Config2__ClassificationHierarchy__c WHERE Name =:
categoryName];
        hierarchyIds.add(categoryList[0].ID);
    }

    ID apexBatchJobId = Apttus_CpqApi.BatchUpdateService.
updateCategoryViews(hierarchyIds);

    ApexPages.addMessage(new ApexPages.Message(ApexPages.severity.
info, 'Apex Batch Job Id : ' + apexBatchJobId));
}
```

Creating Favorite Configuration

This API creates a favorite configuration from the Catalog page referenced by cartID.

saveFavoriteConfiguration

Parameters		
Name	**Type**	**Description**
request	FavoriteConfigurationStruct.SaveFavoriteRequestDO	The request data object.

Request Data Object - FavoriteConfigurationStruct.SaveFavoriteRequestDO		
Field	**Type**	**Description**
cartID	ID	The identifier of the cart.

Request Data Object - FavoriteConfigurationStruct.SaveFavoriteRequestDO		
Field	**Type**	**Description**
favoriteConfigurationSO	Object	Reference to the Favorite Configuration Object
lineNumbers	List <Integer>	The list of line numbers to be added to the favorite configuration.

Data Object - favoriteConfigurationSO		
Field	**Type**	**Description**
Name	String	Enter a suitable name for your favorite configuration. This name is displayed as a record under your Favorites category.
Scope	String	Choose an option for the visibility of your configuration record. The available values are, *Public* and *Private*. If you set this to *Private*, this record is visible to only the owner of the record. If you set this to *Public*, this record is accessible to all the users.
PriceListId	String	The pricelist id associated with a favorite configuration.

Response Data Object - SaveFavoriteResponseDO		
Field	**Type**	**Description**
processedLineNumbers	List<Integer>	List of line numbers added as a favorite configuration.

Code Sample

The sample below enables you to create a favorite configuration for a cart with a cartId. You can invoke this API in use cases when you want to show a Favorite configuration page based on the cartID and pricelist.

```
Apttus_Config2.FavoriteConfigurationStruct.SaveFavoriteRequestDO
requestDO = new Apttus_Config2.FavoriteConfigurationStruct.
SaveFavoriteRequestDO();
System.debug('requestDO\'s favorite configuration : ' +
requestDO.favoriteConfigurationSO);
requestDO.favoriteConfigurationSO.Name = 'HG Fav BundleandStd';
requestDO.favoriteConfigurationSO.Apttus_Config2__Scope__c = 'Public'
;
requestDO.favoriteConfigurationSO.Apttus_Config2__PriceListId__c = 'a
ElF0000000Gmhp';
requestDO.cartId = ID.valueOf('a1I4C0000009hYV');
requestDO.lineNumbers = new List<Integer>();
Integer maxLineNumber = 5;
for (Integer i=1; i<=maxLineNumber; i++)
{ requestDO.lineNumbers.add(i); }
System.debug('response = ' + Apttus_Config2.
FavoriteConfigurationGlobalService.saveFavoriteConfiguration
(requestDO));
```

BatchUpdate Web Service

The Batch Update Web Services enable you to update the single and multiple Category Views.

Updating a Specific Category

Using this API enables you to update a category using the hierarchyID of a specific Category Hierarchy. You can invoke this API after you have associated a product or a bundle to the Category.

updateCategoryView

Parameters		
Name	**Type**	**Description**
hierarchyId	string	ID of the Category you want to update.

Response Object		
Field	**Type**	**Description**
apexBatchJobId	Id	Id of the Apex Batch job.

Code Sample

The sample below enables you to update the Category based on the hierarchyID of the category. Using the sample code below you can fetch the categoryId using the categoryName.

```
public void updateCategoryView(String categoryName)
{
    List<Apttus_Config2__ClassificationHierarchy__c> categoryList =
[SELECT ID FROM Apttus_Config2__ClassificationHierarchy__c WHERE
Name =: categoryName];

    String hierarchyId = categoryList[0].ID;

    ID apexBatchJobId = Apttus_CpqApi.BatchUpdateService.
updateCategoryView(hierarchyId);

    ApexPages.addMessage(new ApexPages.Message(ApexPages.severity.
info, 'Apex Batch Job Id : ' + apexBatchJobId));
}
```

Updating Multiple Categories

Using this API enables you to update multiple categories using a set of hierarchyIDs of multiple Category Hierarchy. You can invoke this API after you have associated a product or a bundle to the Category.

updateCategoryViews

Parameters		
Name	**Type**	**Description**
hierarchyId	set	Set of the hierarchyIds of the categories you want to update.

Response Object		
Field	**Type**	**Description**
apexBatchJobId	Id	Id of the Apex Batch job.

Code Sample

The sample below enables you to update multiple categories based on hierarchyIDs of categories. Using the sample code below you can fetch multiple categoryId using the categoryName.

```
public void updateCategoryViews(List<String> categoryNames)
{
    Set<ID> hierarchyIds = new Set<ID>();

    for(String categoryName : categoryNames) {

        List<Apttus_Config2__ClassificationHierarchy__c>
categoryList = [SELECT ID FROM
Apttus_Config2__ClassificationHierarchy__c WHERE Name =:
categoryName];

        hierarchyIds.add(categoryList[0].ID);
    }

    ID apexBatchJobId = Apttus_CpqApi.BatchUpdateService.
updateCategoryViews(hierarchyIds);

    ApexPages.addMessage(new ApexPages.Message(ApexPages.severity.
info, 'Apex Batch Job Id : ' + apexBatchJobId));
}
```

Registering Split Cart Parameters

The API *setSplitCartInfo* is used to group the line items in the cart for pricing. You can define the parameters, namely the threshold and the split criteria based on which you want to split the line items in the cart through the API. If you do not define the parameters, the values of settings **Split Cart Threshold** and **Split Cart Criteria Fields** in Config System Properties are used to split the line items instead.

setSplitCartInfo

Parameters			
Name	**Type**	**Required?**	**Description**
cartId	ID	Yes	The id of the Cart you want to split.
splitInfo	CustomClass.SplitCartInfo	Yes	The request object.

Request Object - CustomClass.SplitCartInfo		
Field	**Type**	**Description**
SplitCriteriaFields	List	The list of API names of the fields to use as criteria.
SplitThreshold	Integer	The number of line items in a group.

Code Sample

The sample below enables you to set the values of **Split Cart Threshold** and **Split Cart Criteria Fields** to define the information required to split a cart.

```
//Setting Split info
Apttus_Config2.CustomClass.SplitCartInfo splitInfo = new
Apttus_Config2.CustomClass.SplitCartInfo();
splitInfo.SplitCriteriaFields.add('Apttus_Config2_LocationId_c');
splitInfo.SplitThreshold = 10;
//Calling SplitInfo
ID cartID = 'a1I2f00000029Iy'; // cartID can be Main Cart Id or
Config Cart Id
Apttus_Config2.CPQWebService.setSplitCartInfo(cartID , splitInfo);
```

Quote/Proposal Config Web Service

The Quote/Proposal Config web service API account for the standard actions to configure, price, and quote.

You can use the Quote/Proposal Config web service API to complete the following tasks:

- Accepting a Quote

Accepting a Quote

This API accepts a quote or proposal.

acceptQuote

Parameters			
Name	**Type**	**Required?**	**Description**
QuoteID	ID	Yes	The id of the quote/proposal you want to accept.

Response Data Object - CPQ.CreateCartResponseDO		
Field	**Type**	**Description**
isSuccess	Boolean	Indicates whether the quote is accepted or not. The ID of the newly created cart object

Code Sample

The following sample enables you to accepts quote with a Quote ID. Provide a Quote ID of the Quote or Proposal you want to accept. If the quote is successfully accepted, the API returns true, otherwise, the API returns false.

```
/**
  * Accepts the given quote/proposal
  * @param quoteOrProposalId the id of the quote/proposal sobject to
accept
  * @return <code>true</code> if the operation was successful,
<code>false</code> otherwise

 */

    WebService static Boolean acceptQuote(ID quoteOrProposalId) {
        Id quoteOrProposalId = 'a0Y3C000000u2xv';
```

```
        Boolean isSuccess = Apttus_QPConfig.QPConfigWebService.
acceptQuote(quoteOrProposalId);
    }
```

Collaboration Structure

The Collaboration Structure global method account for the standard action to Quote Collaboration feature.

Global methods are not like APIs. They can only be invoked in an Apex Code.

In this Section:

- Adding Products to a Collaboration Request

Adding Products to a Collaboration Request

This method adds products to an existing collaboration request. This global method is not an API. It can only be invoked in an Apex Code.

addProductsToCollaboration

Parameters			
Name	**Type**	**Required?**	**Description**
request	CollabStruct. AddProductCRRequestDO	Yes	The request data object.

Request Data Object - CollabStruct.AddProductCRRequestDO		
Field	**Type**	**Description**
collaborationRequestId	ID	The id of the collaboration request to which you want add products.
lineItemIds	List	The list of line item IDs you want to add to the collaboration request

Response Data Object - CollabStruct.AddProductCRResponseDO		
Field	**Type**	**Description**
errorMessages	List	The list of errors occurred while adding products to the collaboration request.
hasErrors	Boolean	Indicates that the errors occurred while adding products to the collaboration request.
isSuccess	Boolean	Indicated that the products are successfully added to the collaboration request.

Code Sample

The following sample enables you to add products to an existing collaboration request with a Collaboration ID. Provide a list of line item IDs of the products you want to add to the collaboration request. If the products are successfully added to the collaboration request, the API returns true, otherwise, the API returns the list of errors occurred while adding the products.

```
/**
 * The below method demonstrates how to add a product in an existing
collaboration request
 */
    CollabStruct.AddProductCRRequestDO request = new CollabStruct.
AddProductCRRequestDO();
    request.collaborationRequestId = 'a3B1S0000005V3wUAE';
    request.lineItemIds = new List<Id> {'a0a1S000005YwWj'};
    CollabStruct.AddProductCRResponseDO response =
QuoteCollaborationService.addProductsToCollaboration(request);
    if (!response.isSuccess)
    { system.debug(response.errorMessages);
    }
```

Asset Service

The Asset Serivce method accounts for the standard action to Asset-based Operation.

Global method are not APIs. They can only be invoked in an Apex Code.

In this Section:

Incrementing Assets

You can invoke **incrementAssets**, a global method to increment assets.

incrementAssets

Parameters		
Name	**Type**	**Description**
Apttus_Config2.CPQStruct. IncrementAssetRequestDO	request	Request object invoked by the method

Request Data Object - CPQStruct.IncrementAssetRequestDO		
Field	**Type**	**Description**
IncrementAssetDOs	List <Apttus_Config2. CPQStructIncrementAssetDO>	List of increment assets data objects.
CartId	ID	The ID of the cart that consists of assets to be incremented.

CPQStruct.IncrementAssetDO		
Field	**Type**	**Description**

CPQStruct.IncrementAssetDO		
AssetId	Id	The ID of the Asset you want to apply the increment.
LineAction	String	This defines the update made to the asset. There are two types. • *Increment*: creates a new asset • *Increment or* Merge: updates the existing asset.
NewStartDate	Date	New start date for the updated or new asset.
NewEndDate	Date	New end date for the updated or new asset.
Quantity	Integer	The amount of quantity to be incremented.

CPQStruct.IncrementAssetsResponseDO		
Field	**Type**	**Description**
LineItemMap	Map<ID, Apttus_Config2__LineItem__c>	Returns all line items with all their field values.

Code Sample

The following sample enables you to increment the quantity of standalone products. You can also provide new start and end dates for the products. If you define **LineAction** as *Increment*, new assets are created and, if you define **LineAction** as *Increment and Merge*, existing assets are updated.

```
Apttus_Config2.CPQStruct.IncrementAssetRequestDO request = new
Apttus_Config2.CPQStruct.IncrementAssetRequestDO();

for (AssetLineItemWrapperClass record : wrapperAssetLineItemList)
{
     if (record.selected)
    {
          // create and populate request object
```

```
            Apttus_Config2.CPQStruct.IncrementAssetDo assetDO = new
Apttus_Config2.CPQStruct.IncrementAssetDO();
            assetDO.AssetId = record.assetId;
            assetDO.NewStartDate = record.objAssetLineItem.
Apttus_Config2__StartDate__c;
            assetDO.NewEndDate = record.objAssetLineItem.
Apttus_Config2__EndDate__c;
            assetDO.Quantity = record.objAssetLineItem.
Apttus_Config2__Quantity__c;
            assetDO.LineAction = 'Increment'; // Or 'Increment and
Merge'
            request.IncrementAssetDOs.add(assetDO);

        }

}
request.CartId = cartId;

// call incrementAssets API
Apttus_Config2.CPQStruct.IncrementAssetsResponseDO response =
Apttus_Config2.AssetService.incrementAssets(request);
ApexPages.addMessage(new ApexPages.Message(ApexPages.severity.info,
incrementAssets: ' + response));
```

Batch Job Service

The Batch Job Service APIs account for the standard actions to configure, price, and quote.

You can use the Batch Job Service APIs to complete the following tasks:

- Splitting a Smart Cart
- Updating Price for the Smart Cart

Splitting a Smart Cart

This API splits the line items in the cart into groups in a Smart Cart flow.

splitCart

Request Parameter			
Name	**Type**	**Required?**	**Description**
cartID	ID	Yes	The ID of the cart for which you want to split.

Response Parameter		
Field	**Type**	**Description**
jobId	ID	The ID job executed to split the cart.

Code Sample

This sample enables you to splits the line items in the cart into groups in a Smart Cart flow. When the API is executed, the task pending flag is set to true on the cart passed as the parameter. After the line items are split into groups, the flag is set to false.

```
//Api to split large cart (pass in either Main Cart Id or Config
Cart Id). This would set the task pending flag on the cart passed as
the parameter (either main or config). When the split is completed,
the flag is cleared. Note that it will call the pricing once split
is done.

ID jobId = Apttus_Config2.BatchJobService.splitCart
(mainOrConfigCartId);
```

Updating Price for the Smart Cart

This API updates the prices of the line items after the cart is split.

updatePriceForCart

Request Parameter			
Name	**Type**	**Required?**	**Description**
cartID	ID	Yes	The ID of the cart for which you want to update prices of the line item

Response Parameter		
Field	**Type**	**Description**
jobId	ID	The ID batch job executed to update the pricing of the cart.

Code Sample

This sample enables you to update the price of the line items after the cart is split. When the API is executed, the task pending flag is set to true on the cart passed as the parameter. After pricing is calculated for the line items, the flag is set to false.

```
//Api to price large cart (pass in either Main Cart Id or Config
Cart Id), if there is a change in any pricing parameter post split.
This would set the task pending flag on the cart passed as the
parameter (either main or config). When all carts are priced, the
flag is cleared.

ID jobId = Apttus_Config2.BatchJobService.updatePriceForCart
(mainOrConfigCartId);
```

Scenarios

- Working with Products in the Shopping Cart
  After you set up the user roles and use either sites or communities to build the basic framework for your cart page, create a visual force page and a sample CSS to define the look and feel for your cart. You can then proceed with using the Apttus APIs to create a working catalog page.

Working with Products in the Shopping Cart

After you set up the user roles and use either sites or communities to build the basic framework for your cart page, create a visual force page and a sample CSS to define the look and feel for your cart. You can then proceed with using the Apttus APIs to create a working catalog page.

- Searching Products
- Configuring and Adding Products
  In the product catalog, you can see the list of products that a vendor offers. Using this list, you can add a product to your cart, configure the product attributes, such as color and memory. If the product is a bundled product, you can select the options with which you want to bundle your product. This section lists the scenarios using which the customer can view the configuration options for a product.
- Displaying Recommendations
  In the product catalog, you can see the list of products that a vendor offers. When the customer selects a product, you can display the list of recommended products on the same page . This section lists the scenarios using which the recommended products appear. To configure recommendations for a product use the following APIs:
- Updating Price
- Viewing Price Break-up Details
- Adding Price Ramps to a Cart
  After you add a line item to a cart, this API enables you to add primary and secondary ramp line items for the line item. Once the ramp line items are created, you can also update the ramp line item details or delete ramp line item details using standard SOQL queries.

- **Viewing Configured Order in the Shopping Cart**
  After the customer has configured and added all the products to the cart, display the detailed shopping cart items. This section lists all the scenarios using which the customer can view the finalized shopping cart.
- Updating Taxes and Shipping for an Order
- Creating and Updating Quote and Quote Line Items
- Creating and Updating Order and Order Line Items
- Asset Based Ordering
  The end user or a renewals manager can view and modify products that the organization owns and should be able to filter the assets to renew orders. Display and modify the assets that belong to a company based on the following criteria:
- Finalizing the Cart
- Updating Quote Terms to Modify Dates

Searching Products

A customer browsing a product catalog can navigate for a product and select the required product manually or can search for a product. You can enable searches using the search text field in the cart page and enable the user to search for a desired product by category, sub-category, or options available with the product. This section lists all the possible scenarios for which a customer can search or view a product listing.

To browse products

1. Create a cart using Creating a Cart from a Quote.
2. Browse products by categories and subcategories using the search text entered by the user by utilizing Retrieving Products and List Prices For a Price List Category and Search Text and Retrieving Products and List Prices For a Price List and Search Text. Using this API, the customer can search for a product directly or select a category and enter search text to refine the search further.
3. Browse for categories and option groups using Retrieving Categories for a Price List and Retrieving Option Groups, Options, and List Prices for a Price List Product.
4. Browse for products using Retrieving Products and List Prices for a Price List and Retrieving Products and List Prices for a Price List and Category.
5. Compare the displayed products using Comparing Products.

Configuring and Adding Products

In the product catalog, you can see the list of products that a vendor offers. Using this list, you can add a product to your cart, configure the product attributes, such as color and memory. If the product is a bundled product, you can select the options with which you want to bundle your product. This section lists the scenarios using which the customer can view the configuration options for a product.

When you configure and add products to your cart, you can do the following:

- Show Product / Bundle Details
- Create Cart (Create Quote first and then Cart for the Quote)
- Configure Bundles
- Add Single Product to Cart
- Add Multiple Products to Cart
- Show Recommendations for Selected Product
- Show Validation and other messages
- Auto-Include Products
- Delete single product from Cart
- Delete Multiple products from Cart
- Apply any constraint rules related to Cart
- Remove any constraint rules for a deleted product.

To add products and apply constraint rules in the cart

1. Create a cart using createCart.
2. Add products and options to the cart using addBundle , addMultiProducts, or Adding Options to a Bundle.
3. Associate constraint rules for the added products, by querying the right set of rules, using associateConstraintRules.
4. Apply constraint rules to the products in the cart using applyConstraintRules.
5. View the results of applying the constraint rules using getConstraintRuleResult. If the results return *NeedMoreProcessing*, then applyConstraintRules must be run again.

To delete products and apply constraint rules for deleted line items

1. Delete products from the cart using removeBundle or Removing Multiple Bundles from a Cart
2. Remove constraint rules for a deleted product using Applying Constraint Rules to Deleted Products
3. View the results of removing the constraint rules using getConstraintRuleResult.

Displaying Recommendations

In the product catalog, you can see the list of products that a vendor offers. When the customer selects a product, you can display the list of recommended products on the same page . This section lists the scenarios using which the recommended products appear. To configure recommendations for a product use the following APIs:

Scenario 1
After a customer searches a product, the customer selects a product by clicking the product name or image in the result view. The recommendations for that product along with product details should appear.

Scenario 2
After a customer searches a product, the customer adds the product to the cart using the Add to Cart button. On the Cart Details page, the recommendations for that product along with product details should appear along with the cart line items.

For each recommended product, display the product image, name, description, and price.

Updating Price

If you want to update price for a cart after a specific interval use the Updating Price For A Cart API.

You can use the Update Price API for your cart in the following scenarios:

- When the quantity of a product is incremented or decremented.
- When an additional product is added to your cart.
- If any additional charges apply for the product added.
- If any Adjustment whether Markup or Discounts are applied to the cart.

Viewing Price Break-up Details

If the customer wants to view the breakup of how the tiered pricing rules are applied for a given quantity / term of the line item use the Price Breakup for a Cart or Specific Line Item.

The customer might want to see the breakup of how the tiered pricing rules are applied for a given quantity / term of the line item. For example:

There is a tiered pricing structure for a given quantity as follows:

- 0-20 - 5% Discount
- 21-30 - 10% Discount
- > 30 - 15% Discount

If the line item has 25 quantities then show the pricing breakup as

- Product 1 - List Price: $10 Quantity: 20 Discount: 5% Net Price: $190
- Product 1 - List Price: $10 Quantity: 5 Discount: 10% Net Price: $180

Tiered pricing is applicable in the following scenarios:

- Tiered Pricing for a Single Larger Tiers
    - Tiered Pricing defined in Price List
    - Tiered Pricing negotiated in the quote
    - Tiered pricing negotiated in pricing agreement
- Tiered Pricing for a Single Order leading to higher tier e.g. single order of quantity 25 in above case
- Tiered Pricing for new and add-on order leading to higher tier e.g. initial order of 15 and then add-on (increment) asset based order of 10 quantity
- Tiered for different price types
    - Usage Tiers
    - One time Charge Tiers
    - Recurring subscription tiers
- Ramps

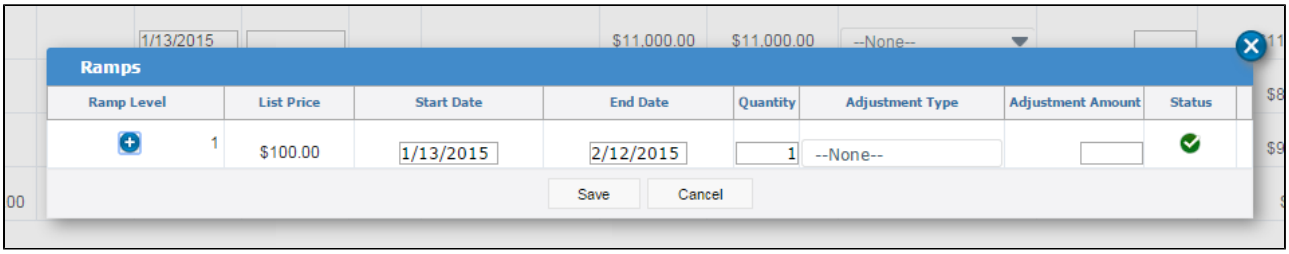
The breakup details for each line item should display a cart id, line id, and price break up details, such as quantity tier, list price, base price, extended price, adjustment, and net price.

Adding Price Ramps to a Cart

After you add a line item to a cart, this API enables you to add primary and secondary ramp line items for the line item. Once the ramp line items are created, you can also update the ramp line item details or delete ramp line item details using standard SOQL queries.

When you use Apttus CPQ out of the box, invoke the ramp using the red icon to the left of the primary line item.

Once you click the ramp icon, the ramp dialog appears:



The ramp dialog allows you to add, edit dates and quantity, make adjustments, save the changes, and cancel the changes.

Once the customer adds a ramp to a cart line item, they can do the following:

- Edit the start date, end date, quantity, adjustment type and adjustment amount based on the custom setting.
- The start date of a ramp line item defaults to the end date+1 of the previous line item.
- The end date of a ramp line item defaults to a date such that the difference between the start date and end date is the same as that of the previous line item.
- The user can add more ramp line items after or in between the ramp line items.
- The user can remove the new line before saving by clicking on the icon in the right most column.

Use the addMultiProducts API to add products to the cart.

addMultiProducts

Parameters		
Name	**Type**	**Description**
request	CPQ.AddMultiProduct RequestDO	The request data object.

Request Data Object - CPQ.AddMultiProductRequestDO		
Field	**Type**	**Description**
CartId	ID	The Id of the cart.
SelectedProducts	List <CPQ. SelectedProductDO>	The list of selected product data objects.

Response Data Object - CPQ.AddMultiProductResponseDO		
Field	**Type**	**Description**
LineNumbers	List<Decimal>	The list of line numbers added to the cart.

Data Object - CPQ.SelectedProductDO		
Field	**Type**	**Description**
ProductId	ID	Id of the product bundle.
Quantity	Decimal	The bundle quantity.
SellingTerm	Decimal	The bundle selling term.
StartDate	Date	The start date. You should ensure you use the correct date format.

Data Object - CPQ.SelectedProductDO		
Field	**Type**	**Description**
EndDate	Date	The end date.
Comments	String	Comments associated with the record.
CustomFields	List<String> CustomFields	List of custom fields created for your product.
CustomData	Apttus_Config2__LineItem__c CustomData	This can be used to include the list of custom fields you have added to the product.

Code Sample

The sample below enables you to add ramp line items after you have:

- Added Products to the cart using the AddMultiProducts APIs,
- Updated the Price for the added products using the updatePriceforCart API.
- Selected the products for which you want to add a ramp for.

Using the sample below you fetch the list of selected products for which you want to add a ramp. You also fetch the parameters for each of the selected products. For all the ramps you create, set PriceGroup as Price Ramp and PricingStatus as Pending. For a primary line item, set IsPrimaryLine__c = true, IsPrimaryRampLine__c = true, and PrimaryLineNumber__c = 1.

```
public void createRampLineItems()
    {
        List<String> rampLineItems = new List<String>();

        if(lstWrapItems.size() > 0)
        {
            // Create a list of selected products for which you want
to create a ramp for
            for(LineItemWrapperClass objLineItemWrapperClass :
lstWrapItems)
            {
                if(objLineItemWrapperClass.Selected)
                {
                    rampLineItems.add(objLineItemWrapperClass.Name);
```

```
                }
            }

            // Sort Ramp Line Items by name
            rampLineItems.sort();

            Integer rampLineItemIndex = 1;

            for(String rampLineItemName : rampLineItems)
            {
                // Get Line Item parameters for the selected products
                Apttus_Config2__LineItem__c lineItem = [SELECT
Apttus_Config2__ItemSequence__c, Apttus_Config2__PricingStatus__c,
Apttus_Config2__PriceGroup__c,
                Apttus_Config2__IsPrimaryRampLine__c,
Apttus_Config2__IsPrimaryLine__c, Apttus_Config2__LineNumber__c,
Apttus_Config2__PrimaryLineNumber__c from
Apttus_Config2__LineItem__c WHERE
                Name=:rampLineItemName];

                //Set the parameters for each of the line items
                lineItem.Apttus_Config2__PriceGroup__c = 'Price Ramp'
;
                lineItem.Apttus_Config2__PricingStatus__c = 'Pending'
;
                lineItem.Apttus_Config2__LineNumber__c = 1;
                lineItem.Apttus_Config2__PrimaryLineNumber__c = 1;
                lineItem.Apttus_Config2__ItemSequence__c =
rampLineItemIndex;
                //For a primary line item set the following
            `    if(rampLineItemIndex == 1)
                {
                    lineItem.Apttus_Config2__IsPrimaryLine__c = true;
                    lineItem.Apttus_Config2__IsPrimaryRampLine__c = t
rue;
                }

                //For all secondary line items set the following
parameters
                else
                {
                    lineItem.Apttus_Config2__IsPrimaryLine__c = false
;
                    lineItem.Apttus_Config2__IsPrimaryRampLine__c = f
alse;
                }
```

```
                // Update Line Items
                update lineItem;

                rampLineItemIndex++;
            }
        }
        else
        {
            ApexPages.addMessage(new ApexPages.Message(ApexPages.
severity.info, 'No line items available.'));
        }
    }
```

Updating Taxes and Shipping for an Order

After the user has added products to the cart, you apply taxes to each of the cart line items and shipping charges for the entire order.

- Tax and shipping can be calculated for the Quotes (Estimated tax / Shipping amount), but will be more generally used with E-Commerce orders with Pay Now scenario as the actual tax and shipping amount.

After the user has added the products to the cart, the sequence of actions is as follows:

- Customer specifies the Ship-to location.
- Once the ship to account or location is specified at the header (line item level in case of multiple ship-to locations)level, invoke the APIs to calculate Shipping as well as tax.
- Once the shipping and tax charges are calculated, the total price, shipping, and tax amount is displayed for a customer to review.
- Customer will then specify Payment option and Place the order.

Tax Calculation Workflow:

- Create Tax callback class.
- Implement tax interface ITaxCallback2 in the callback class to call the tax engine to calculate the total tax for the cart (Quote / Order) and breakup for each line item.
- Invoke computeTaxforCart API to which we just pass the “CartId”. The API will create the tax breakup line items as well as the total tax line for the entire cart.

Shipping Callback Workflow:

- Create Shipping callback class.

- Implement shipping interface IShippingCallback in the callback class to call the shipping engine to calculate the total shipping charges for the cart (Quote / Order).
- Invoke computeShippingForCart to which we just pass the “CartId”. The API will create the shipping charges for the entire cart.
- Sample Callback for Calculating Shipping
  This section comprises the sample callback class that enables you to apply an appropriate shipping charge to a cart.
- Sample Callback for Calculating Taxes
  This section comprises the sample callback class that enables you to apply an appropriate tax to a cart line item.
- To apply taxes and shipping charges

Sample Callback for Calculating Shipping

This section comprises the sample callback class that enables you to apply an appropriate shipping charge to a cart.

Before you define the shipping callback ensure that you have set the following:

- If shipping charges change based on the ship to or shipping address of the account, ensure that the account associated with the proposal has both specified.

  In addition, you can add additional custom fields on the quote or order line item such as Shipping Speed to calculate the shipping charge.

The sample callback class comprises the following objects:

Data Object - Apttus_Config2.ShippingInput		
Field	**Type**	**Description**
ShippingAddress	Address	Shipping Address. The address for shipping charge calculation obtained from the "Ship To" account for the line item. Accordingly, Ship To Account should be populated on each line item by the implementation.
Item	SObject	

Data Object - Apttus_Config2.ShippingInput		
Field	**Type**	**Description**
		Apttus_Config2__LineItem__c. The Cart Line item. This is the reference to the cart line items. Implementation can use this cart line item to get any additional custom fields that might be needed to calculate Shipping.
Handback	Object	Any Apex Object passed by the caller and passed back to the caller in the result. Handback is used only in case when the shipping needs to be calculated for an object other than Cart line item. For example, Billing Schedule line items. • For calculating shipping for cart , there is no need to pass the Handback object. It can be blank • For calculating the Shipping for object other than line items, "Item" field will be blank and the actual object will be passed as handback object to identify object for which shipping is calculated.

Data Object - Apttus_Config2.CustomClass.ShippingResult		
Field	**Type**	**Description**
ShippingCharge	Decimal	Shipping amount total of all the cart line items. This is the total shipping amount for the line item. Line item is identified by the sequence in which the shipping result is added. This needs to be calculated and populated by the implementation.
Handback	Object	Any Apex Object passed by the calling class and passed back to the calling class in the result. Handback is used only in case when the shipping charge needs to be calculated for an object other than Cart line item e.g. Billing Schedule line items.

Data Object - Apttus_Config2.CustomClass.ShippingResult		
Field	**Type**	**Description**
		• For calculating shipping charges for a cart, there is no need to pass the Handback object. It can be blank • For calculating the shipping charges for objects other than line items, "Item" field will be blank and the actual object will be passed as handback object to identify the object for which shipping charges are calculated.

Once you have set shipping to, shipping address or shipping speed to your quote or order, create a Shipping Callback class and specify the name of the callback class in **Custom Settings > Config Custom Classes > (Edit) Custom Classes > Shipping Callback Class**.

In the sample callback class below, we fetch and create a list of all the products added to the cart. Then we fetch the shipping charges applicable for a product using an SOQL query. If a corresponding shipping code and a valid shipping charge exist for a product in the order or cart, the total shipping amount for a cart is added directly.

Sample Callback for Calculating Taxes

This section comprises the sample callback class that enables you to apply an appropriate tax to a cart line item.

Before you define the tax callback ensure that you have set the following:

- Go to the product price list item, click **Edit > Tax & Billing** and select the **Taxable?** checkbox. Also ensure that the **Tax Inclusive?** checkbox is deselected. If it is selected, the product price specified includes the tax amount.
- From the **Tax&Billing** tab, specify the tax code applicable for a product using a look up. The tax code maps to a code in an external system where various taxes, tax charges, tax types are mapped against a tax code.
- If a tax is applicable based on the billing or shipping address of the account, ensure that the account associated with the proposal has both specified.

The sample callback class comprises the following objects:

Data Object - Apttus_Config2.TaxInput		
Field	**Type**	**Description**
TaxCode	String	Tax code for the line item as inherited from the Price list item or overridden at the line item level.
TaxAddress	Address	Tax Address. Address as obtained for the line item based on precedence hierarchy specified. For example, Line Item Location ==> Line Item Ship To ==> Line Item Bill To ==> Cart Header Ship To ==> Cart Header Bill To ==> Cart Header Sold to
TaxableAmount	Decimal	Net Price from the Line Item in case the tax needs to be calculated on the Net Price. In case the tax needs to be calculated based on List or Base Price then the total amount calculated based on list or base price needs to be used from the line item to calculate tax.
Item	SObject	Apttus_Config2__LineItem__c. This is the reference to the cart line items. Implementation can use this cart line item to get any additional custom fields that might be needed to calculate tax.
Handback	Object	Any Apex Object passed by the caller and passed back to the caller in the result. Handback is used only in case when the tax needs to be calculated for an object other than Cart line item e.g. Billing Schedule line items. For calculating tax for cart line items, there is no need to pass the Handback object. It can be blank. For calculating the tax for object other than line items, "Item" field will be blank and the actual object will be passed as handback object to identify object for which tax is calculated.

Data Object - Apttus_Config2.CustomClass.TaxResult		
Field	**Type**	**Description**
TaxAmount	Decimal	This is the total amount for the line item. This is sum of the breakup tax amounts for the line item. Line item is identified by the sequence in which the tax result is added. This needs to be calculated and populated by the implementation.
TaxBreakups	List<Apttus_Config2__TaxBreakup>	Tax breakup provide the different components of the tax for a given line item. This needs to be implemented and provided by the implementation. Tax breakup has structure specified below.
Handback	Object	Any Apex Object passed by the calling class and passed back to the calling class in the result. Handback is used only in case when the tax needs to be calculated for an object other than Cart line item e.g. Billing Schedule line items. • For calculating tax for cart line items, there is no need to pass the Handback object. It can be blank • For calculating the tax for object other than line items, "Item" field will be blank and the actual object will be passed as handback object to identify object for which tax is calculated.

Data Object - Apttus_Config2__TaxBreakup		
Field	**Type**	**Description**
TaxType	String	Type of tax to be applied. For example, sales, custom, excise
TaxRate	Decimal	Tax percentage to be applied
TaxAppliesTo	String	The field on which the tax is applied to. For example, net price.
TaxAmount	Decimal	The tax amount calculated based on the formulae defined in the callback.
BreakupType	String	If you specify the breakup type as Detail, then the quote or order line items show the detailed tax break up for a cart line item.

Tax Breakup Type Example

Breakup Type	Tax Type	Tax Rate	Tax Applies to	Tax Amount
Detail	State Tax	10	Net Price	100
Detail	City Tax	1	Net Price	10
Total				110

Data Object - Apttus_Config2. CustomClass.Address	
Field	**Type**
Street	String
City	String
State	String

Data Object - Apttus_Config2. CustomClass.Address	
Field	**Type**
County	String
PostalCode	String
Country	String

Once you have set a Tax Code for each price list item, create a Tax Callback class and specify the name of the callback class in **Custom Settings > Config Custom Classes > (Edit) Custom Classes > Tax Callback Class**.

In the sample callback class below, we fetch and create a list of all the products added to the cart. Then we fetch the tax charges applicable for a product using an SOQL query. If a corresponding tax code and a valid tax charge exists for a product in the order or quote line item, the total tax amount for a product is calculated using the formula- taxAmount += (taxRate != null ? (taxRate * input.TaxableAmount) / 100 : 0). After the total tax on a product is calculated, you can list the tax break up based on Tax types applied for each of the order or quote line items. The sample callback lists the parameters applicable for a tax break up. If you want to apply tax to shipping charges as well, you can capture shipping amount for each line item in a custom field and use the total amount (Net Price + Shipping Amount) from the line item to calculate tax. In this case shipping should be calculated first and then tax.

In the sample callback class, tax is calculated using hardcoded rates. For actual implementations, tax rates and tax amounts are fetched from external Tax calculation engines such as Avalara and Vertex to calculate tax.

```
/**
 *  Apttus Config & Pricing
 *  QATaxCallBack
 *
 *  @2013-2014 Apttus Inc. All rights reserved.
 */
global with sharing class QATaxCallBack implements Apttus_Config2.
CustomClass.ITaxCallback2 {

    /**
     * Callback invoked to compute tax based on the given input
```

```
     * @param input the tax input
     * @return the tax result
     */
    global Apttus_Config2.CustomClass.TaxResult computeTax
(Apttus_Config2.CustomClass.TaxInput input) {
      // Compute Tax
      system.debug('Entering into computeTax');
      List<Apttus_Config2.CustomClass.TaxResult> results =
computeTaxMultiple(new Apttus_Config2.CustomClass.TaxInput[]
{input});
      return (!results.isEmpty() ? results[0] : new Apttus_Config2.
CustomClass.TaxResult());
    }

    /**
     * Callback invoked to compute tax based on the given list of
inputs
     * @param inputs the list of tax inputs
     * @return the list of tax results
     */
    global List<Apttus_Config2.CustomClass.TaxResult>
computeTaxMultiple(List<Apttus_Config2.CustomClass.TaxInput> inputs)
{

        List<Apttus_Config2.CustomClass.TaxResult> results = new
List<Apttus_Config2.CustomClass.TaxResult>();

        for (Apttus_Config2.CustomClass.TaxInput input : inputs) {

            Decimal taxAmount = 0;

            Apttus_Config2.CustomClass.TaxResult result = new
Apttus_Config2.CustomClass.TaxResult();

            List<Apttus_Config2__TaxBreakup__c> taxBreakUps = new
List<Apttus_Config2__TaxBreakup__c>();

            // Add the handback object to correlate the result with
the input
            result.Handback = input.Handback;

            // Compute tax based on tax code and tax address

            // Tax Code is required
            if (input.TaxCode == null || input.TaxCode.trim().
length() == 0 || input.TaxableAmount == null || input.TaxableAmount
== 0) {
```

```
            // Tax Amount
            result.TaxAmount = 0;
          } else {

              List<QA_Tax_Rate__c> accountTaxRates = new
List<QA_Tax_Rate__c>();

              SObject sObj = input.Item;
              Id taxCodeId = (Id) sObj.get('Apttus_Config2__TaxCode
Id__c');

              accountTaxRates =  [SELECT Id, Name,
Tax_Applies_To__c, Tax_Rate__c, Tax_Type__c
              FROM QA_Tax_Rate__c
              WHERE Tax_Code__c =:  taxCodeId];

              for(QA_Tax_Rate__c accountTaxRate : accountTaxRates)
{

                  Decimal taxRate = accountTaxRate.Tax_Rate__c;
                  String taxAppliesTo = accountTaxRate.
Tax_Applies_To__c;
                  String taxType = accountTaxRate.Tax_Type__c;

                  Apttus_Config2.CustomClass.Address addr = input.
TaxAddress;

                  // Calculate Tax Amount
                  taxAmount += (taxRate != null ? (taxRate * input.
TaxableAmount) / 100 : 0);

                  // Calculate Tax BreakUps
                  Apttus_Config2__TaxBreakup__c taxBreakUp = new
Apttus_Config2__TaxBreakup__c();
                  taxBreakUp.Apttus_Config2__TaxType__c = taxType;
                  taxBreakUp.Apttus_Config2__TaxRate__c = taxRate;
                  taxBreakUp.Apttus_Config2__TaxAppliesTo__c =
taxAppliesTo;
                  taxBreakUp.Apttus_Config2__TaxAmount__c =
taxRate * input.TaxableAmount / 100;
                  taxBreakUp.Apttus_Config2__BreakupType__c = 'Deta
il';

                  taxBreakUps.add(taxBreakUp);
              }
```

```
                result.TaxBreakups = taxBreakUps;
                result.TaxAmount = taxAmount;
            }
            results.add(result);
        }
        return results;
    }
}
```

To apply taxes and shipping charges

1. Create a cart using the Creating a Cart from a Quote.
2. Browse products by categories and subcategories using the search text entered by the user by utilizing Retrieving Products and List Prices For a Price List Category and Search Text and Retrieving Products and List Prices For a Price List and Search Text. Using this API, the customer can search for a product directly or select a category and enter search text to refine the search further.
3. Browse for categories and option groups using Retrieving Categories for a Price List and Retrieving Option Groups, Options, and List Prices for a Price List Product.
4. Browse for products using Retrieving Products and List Prices for a Price List and Retrieving Products and List Prices for a Price List and Category.
5. Compare the displayed products using Comparing Products.
6. Add products to the cart by using Adding Products to a Cart and Adding a Bundle to a Cart.
7. Update the price using Updating Price For A Cart.
8. Finalize the cart using Finalizing a Cart's Contents.
9. Define the Tax and Shipping Callbacks using Sample Callback for Calculating Taxes and Sample Callback for Calculating Shipping.
10. Specify the Callback class names at **Custom Settings > Config Custom Classes (Edit) > Tax Callback and Shipping Callback**.
11. Apply the taxes and shipping charges using Computing Taxes for Cart Line Items and Computing Shipping for Cart Line Items. The Tax API creates and populates - the tax breakup for each applicable line item in the cart and the total Tax line and related breakup. The shipping API calculates the shipping amount for the entire order.
12. Update the Order Line Items using Synchronizing a Cart.

Creating and Updating Quote and Quote Line Items

A customer with a valid account can create a quote from an existing opportunity. Using the Quote ID and a valid price list, you can create a cart from scratch and by browsing the product catalog you can navigate for a product and select the required product manually or can search for a product. You can enable searches using the search text field in the cart page and enable the user to search for a desired product by category, sub-category, or options available with the product. This section lists all the possible scenarios for which a customer can create a quote and or view a product listing.

To create and update quote and quote line items

1. Create a cart using the Creating a Cart from a Quote.
2. Browse products by categories and subcategories using the search text entered by the user by utilizing Retrieving Products and List Prices For a Price List Category and Search Text and Retrieving Products and List Prices For a Price List and Search Text. Using this API, the customer can search for a product directly or select a category and enter search text to refine the search further.
3. Browse for categories and option groups using Retrieving Categories for a Price List and Retrieving Option Groups, Options, and List Prices for a Price List Product.
4. Browse for products using Retrieving Products and List Prices for a Price List and Retrieving Products and List Prices for a Price List and Category.
5. Compare the displayed products using Comparing Products.
6. Add products to the cart by using Adding Products to a Cart and Adding a Bundle to a Cart.
7. Update the price using Updating Price For A Cart.
8. Finalize the cart using Finalizing a Cart's Contents.
9. Update the Quote Line Items using Synchronizing a Cart.

Creating and Updating Order and Order Line Items

A customer with a valid account can create an order from an existing opportunity. Using the order ID and a valid price list, you can create a cart from scratch and by browsing the product catalog you can navigate for a product and select the required product manually or can search for a

product. You can enable searches using the search text field in the cart page and enable the user to search for a desired product by category, sub-category, or options available with the product. This section lists all the possible scenarios for which a customer can create a order and or view a product listing.

To create and update order and order line items

1. Create an order using Creating an Order
2. Create a cart for an order using Creating a Cart for an Order.
3. Browse products by categories and subcategories using the search text entered by the user by utilizing Retrieving Products and List Prices For a Price List Category and Search Text and Retrieving Products and List Prices For a Price List and Search Text. Using this API, the customer can search for a product directly or select a category and enter search text to refine the search further.
4. Browse for categories and option groups using Retrieving Categories for a Price List and Retrieving Option Groups, Options, and List Prices for a Price List Product.
5. Browse for products using Retrieving Products and List Prices for a Price List and Retrieving Products and List Prices for a Price List and Category.
6. Compare the displayed products using .
7. Add products to the cart by using Adding Products to a Cart and Adding a Bundle to a Cart.
8. Update the price using Updating Price For A Cart.
9. Update the order line items using Synchronizing a Cart to an Order.
10. Create Asset Line Items using Create Asset Line Items for an Order.
    For more information on the Orders APIs, refer to *Order Management on Salesforce Spring 2018 SOAP API Guide*.

Asset Based Ordering

The end user or a renewals manager can view and modify products that the organization owns and should be able to filter the assets to renew orders. Display and modify the assets that belong to a company based on the following criteria:

- All assets with recurring, usage, and one-time charge types for which renewal date is approaching.
- All licenses that expire within 30, 60, or 90 days.
- Expired licenses.

- All services that expire in 30 days.
- All assets that are in the End of Life or End of Sale state by the vendor.
- All services that are no longer active or are cancelled should be displayed.
- All software licenses setup for a site.
- All products under the CPQ product category.

The assets should be filtered and displayed based on asset status, price type, and subscription expiration.

The displayed results of the asset list the product name, ID, asset name, asset ID, SKU ID, quantity, price type, price UOM, purchase date, start and end date, base price, adjustment, net price, asset status, and renewal term.

To view assets

Retrieve asset results that fulfill the criteria using the Retrieving Asset Line Items API. Using this API, you can fetch the assets.

To modify assets

1. Retrieve asset results that fulfill the criteria using the Retrieving Asset Line Items API. Using this API, you can fetch the assets.
2. You can renew, amend, increment, or terminate the assets using the code snippets for Modifying Assets (Deprecated).

To create assets from orders

Create assets from an order using the Create Asset Line Items for an Order API. Using this API, you can create assets.

Finalizing the Cart

The customer can finalize the cart in the following scenarios:

- All the required products have been added.
- All requisite prices have been adjusted accordingly.

Invoke the Finalizing a Cart's Contents to enable the user to finalize the cart.

Updating Quote Terms to Modify Dates

For a finalized quote, a customer can choose to apply add-ons. The term,validity period, and pricing have to be recalculated after the quote is finalized, approved, and a subsequent proposal is generated. Use the Updating Quote Terms to achieve the scenarios highlighted.

The updateQuoteTerms API behavior is as follows:

- Update the carts in Finalized or Ready for Finalization status. The latest cart from the quote in any of the two statuses are used as basis to updating the quote.
- Create a new version of the cart in case of Finalized cart and supersede the current version of the cart. The status of the new version of the cart will be same as the status of the current version of the cart. If the cart was in Finalized status, the new cart will also be in finalized status and system will update the "Finalized date" on the quote header.
- Auto-Synch cart line items with the Proposal Line Items.

To synchronize cart and update quote

1. Finalize the cart using the Finalize Cart API.
2. Synchronize the finalized cart with the quote using the Synchronize API.
3. Update the selling terms of the products associated to the quote using the Update Quote Terms API.

Apttus Copyright Disclaimer

The documentation and/or software may provide links to Web sites and access to content, products, and services from third parties. Apttus is not responsible for the availability of, or any content provided by third parties. You bear all risks associated with the use of such content. If you choose to purchase any products or services from a third party, the relationship is directly between you and the third party. Apttus is not responsible for: (a) the quality of third-party products or services; or (b) fulfilling any of the terms of the agreement with the third party, including delivery of products or services and warranty obligations related to purchased products or services. Apttus is not responsible for any loss or damage of any sort that you may incur from dealing with any third party.

For additional resources and support, please visit https://community.apttus.com.

DOC ID: CPQSFWIN19APIG20191204