Salesforce CPQ Plugins

Version 43.0, Summer '18

@salesforcedocs

Last updated: July 26, 2018

© Copyright 2000–2018 salesforce.com, inc. All rights reserved. Salesforce is a registered trademark of salesforce.com, inc., as are other names and marks. Other marks appearing herein may be trademarks of their respective owners.

CONTENTS

Chapter 1: Salesforce CPQ Plugins 1
Javascript Quote Calculator Plugins 2
Quote Calculator Plugin Methods 3
Calculating True End Date and Subscription Term 4
Custom Package Total Calculation 8
Find Lookup Records 13
Insert Records 16
CPQ Product Search Plugin 17
External Configurator Plugins 26
Set Up an External Configurator to Launch from a Custom Action 26
Set Up a Web Application for Integration 27
Set Up External Configurators for Automatic Launch 27
Page Security Plugin 28
Salesforce CPQ Electronic Signature Plugin 28

Index 30

CHAPTER 1 Salesforce CPQ Plugins

In this chapter ...

- Javascript Quote Calculator Plugins
- CPQ Product Search Plugin
- External Configurator Plugins
- Page Security Plugin
- Salesforce CPQ Electronic Signature Plugin

Salesforce CPQ plugins let you add customized functionality to features within the Salesforce CPQ package.

EDITIONS

Available in: All Salesforce CPQ Editions

Javascript Quote Calculator Plugins

Salesforce CPQ uses its Javascript Quote Calculator to perform calculations within the quote line editor. You can create calculator plugins to add extra functionality.

When you develop a plugin for the Javascript Quote Calculator, consider key guidelines.

EDITIONS

Available in: Salesforce CPQ Winter '16 and later

Promises

A Promise is a built-in JavaScript object that allows for asynchronous programming in the browser. Promises let you delay a certain action until another one has completed. Promises support a `.then(success, failure)` method, where success is a function called when the promise resolves successfully, and failure is a function called when the promise is rejected. If you want to do any asynchronous programming in the plugin, such as a server callout, you must return a promise that resolves once that action is completed. This guarantees that calculation steps occur in the proper order. If a method doesn't require asynchronous behavior, you can return a promise that resolves immediately as `return Promise.resolve();`. Promises can resolve to a value, which passes as a parameter to the `.then()` callbacks. You can use this fact in your own code, but remember that the promises that these methods return don't need to resolve to a value. Always directly modify the quote and line models provided in the parameters.

QuoteModel and QuoteLineModel Types

The JavaScript calculator represents Quote__c and QuoteLine__c objects as QuoteModel and QuoteLineModel objects respectively. You can access the underlying SObject through the `.record` property on both objects, which lets you reference fields by using their API name. For example, you can reference a custom field SBQQ__MyCustomField__c on a given QuoteLineModel by accessing the attribute record ["SBQQ__MyCustomField__c"]. You can also reference fields on related records. For example, if you want to reference the field MyField__c on an account associated with a quote, access the record ["Account__r"]["MyField__c"].

Salesforce Field Types

You can change records stored in JavaScript Object Notation, or JSON. These records are serialized from your org. Number, Text, and Boolean fields are all stored without any conversion, but you can convert any other type. For example, dates are represented as strings of the format "YYYY-MM-DD." If you reference or change a field containing a date, you have to preserve that format.

JSForce

JSForce is a third-party library that provides a unified way to perform queries, execute Apex REST calls, use theMetadata API, or make HTTP requests remotely. Methods access jsforce through the optional parameter `conn`.

Note: The JSQCP is an ES6 module. It is transpiled via Babel and module-scoped by default. You can use any elements of the ES6 language or syntax. However, the plugin must be able to run in both browser and node environments. Global browser variables such as window may not be available.

Quote Calculator Plugin Methods

Javascript Quote Calculator plugins can reference several optional methods. You can export any, all, or none of them to achieve your desired behavior.

Calculating True End Date and Subscription Term

You can use a quote calculator plugin to calculate values and store maximum values for the custom quote line fields True Effective End Date and True Effective Term.

Custom Package Total Calculation

Use a quote calculator plugin to calculate the total price for all components in a quote line and then store that value in the custom field Component Custom Total.

Find Lookup Records

Use a quote calculator plugin to query records within the plugin and use fields from those records to set each quote line's Description field.

Insert Records

Use JSForce in a javascript quote calculator plugin to insert records.

Quote Calculator Plugin Methods

Javascript Quote Calculator plugins can reference several optional methods. You can export any, all, or none of them to achieve your desired behavior.

EDITIONS

Available in: Salesforce CPQ Winter '16 and later

Table 1: onInit

Param	Type	Description
{QuoteLineModel[]}	quoteLineModels	An array containing Javascript representations of all lines in a quote.

The calculator calls this method before formula fields are evaluated. Returns `{promise}`.

```
export function onInit(quoteLineModels) {
return Promise.resolve();
};
```

Table 2: onBeforeCalculate

Param	Type	Description
{QuoteModel}	quoteModel	Javascript representation of the quote you're evaluating
(QuoteLineModel[]}	quoteLineModels	An array containing Javascript representations of all lines in the quote

The calculator calls this method before calculation begins, but after formula fields have been evaluated. Returns `{promise}`.

```
export function onBeforeCalculate(quoteModel, quoteLineModels) {
return Promise.resolve();
};
```

Table 3: onBeforePriceRules

Param	Type	Description
{QuoteModel}	quoteModel	Javascript representation of the quote you're evaluating
(QuoteLineModel[]}	quoteLineModels	An array containing Javascript representations of all lines in the quote

The calculator calls this method before it evaluates price rules. Returns `{promise}`.

```
export function onBeforePriceRules(quoteModel, quoteLineModels) {
return Promise.resolve();
};
```

Table 4: onAfterPriceRules

Param	Type	Description
{QuoteModel}	quoteModel	Javascript representation of the quote you're evaluating
(QuoteLineModel[]}	quoteLineModels	An array containing Javascript representations of all lines in the quote

The calculator calls this method after it evaluates price rules. Returns `{promise}`.

```
export function onAfterPriceRules(quoteModel, quoteLineModels) {
return Promise.resolve();
};
```

Table 5: onAfterCalculate

Param	Type	Description
{QuoteModel}	quoteModel	Javascript representation of the quote you're evaluating
(QuoteLineModel[]}	quoteLineModels	An array containing Javascript representations of all lines in the quote

The calculator calls this method after it completes a calculation, but before re-evaluating formula fields. Returns `{promise}`

```
export function onAfterCalculate(quoteModel, quoteLineModels) {
return Promise.resolve();
};
```

Calculating True End Date and Subscription Term

You can use a quote calculator plugin to calculate values and store maximum values for the custom quote line fields True Effective End Date and True Effective Term.

These are sample Apex and Javascript plugins for use in the JavaScript Quote Calculator. Each version exports all of the methods that the calculator looks for, and documents their parameters and return types.

EDITIONS

Available in: Salesforce CPQ Winter '16 and later

Apex

```
global class QCPWinter16Legacy2 implements SBQQ.QuoteCalculatorPlugin,
SBQQ.QuoteCalculatorPlugin2 {

/* This QCP examples calculates and stores the effective end date on each quote line, as
well as the effective term.
```

```
   It also stores the max(effective end date) and max(effective term) on the Quote object
*/


    /* NOTE: the getReferencedFields method is no longer required if you use the
ReferencedFields field set on
 the Quote Line object.
             This field set must be created as it's not a managed one.
       NOTE: if you need to access Quote fields, you can create the ReferencedFields field
 set on the Quote object as well.
       NOTE: if you do not use the getReferencedFields method, you can remove
SBQQ.QuoteCalculatorPlugin2 from the class declaration.
    */
    global Set<String> getReferencedFields() {
        return new Set<String>{
            /* Note: add fields using the following format - Only add fields referenced

              by the plugin and not in the Line Editor field set on the Quote Line object

            String.valueOf(SBQQ__QuoteLine__c.My_Field_API_Name__c)
            */
            String.valueOf(SBQQ__QuoteLine__c.True_Effective_End_Date__c),
            String.valueOf(SBQQ__QuoteLine__c.True_Effective_Term__c),

            String.valueOf(SBQQ__Quote__c.True_Effective_End_Date__c),
            String.valueOf(SBQQ__Quote__c.True_Effective_Term__c),

            String.valueOf(SBQQ__QuoteLine__c.SBQQ__EffectiveStartDate__c),
            String.valueOf(SBQQ__QuoteLine__c.SBQQ__EffectiveEndDate__c),
            String.valueOf(SBQQ__QuoteLine__c.SBQQ__SubscriptionTerm__c),
            String.valueOf(SBQQ__QuoteLine__c.SBQQ__DefaultSubscriptionTerm__c),

            String.valueOf(SBQQ__Quote__c.SBQQ__SubscriptionTerm__c)
        };
    }

    global void onBeforePriceRules(SObject quote, SObject[] lines) {
    }

    global void onAfterPriceRules(SObject quote, SObject[] lines) {
    }

    global void onBeforeCalculate(SObject quote, SObject[] lines) {
    }

    global void onAfterCalculate(SObject quote, SObject[] lines) {
        Date maxEffectiveEndDate = null;
        Decimal maxEffectiveTerm = 0;
        for(SObject line : lines) {
            Date trueEndDate = calculateEndDate(quote, line);
            Decimal trueTerm = getEffectiveSubscriptionTerm(quote, line);
            if(maxEffectiveEndDate == null || maxEffectiveEndDate < trueEndDate) {
                maxEffectiveEndDate = trueEndDate;
            }
```

```
            if(maxEffectiveTerm < trueTerm) {
                maxEffectiveTerm = trueTerm;
            }
            line.put(SBQQ__QuoteLine__c.True_Effective_End_Date__c, trueEndDate);
            line.put(SBQQ__QuoteLine__c.True_Effective_Term__c, trueTerm);
        }
        quote.put(SBQQ__Quote__c.True_Effective_End_Date__c, maxEffectiveEndDate);
        quote.put(SBQQ__Quote__c.True_Effective_Term__c, maxEffectiveTerm);
    }

    global void onInit(SObject[] lines) {
    }

    private Date calculateEndDate(SObject quote, SObject line) {
        Date startDate =
(Date)line.get(String.valueOf(SBQQ__QuoteLine__c.SBQQ__EffectiveStartDate__c));
        Date endDate =
(Date)line.get(String.valueOf(SBQQ__QuoteLine__c.SBQQ__EffectiveEndDate__c));
        if ((startDate != null) && (endDate == null)) {
            /* Note: we are assuming that Subscription Term Unit is Month in the package
settings */
            endDate = startDate.addMonths(getEffectiveSubscriptionTerm(quote,
line).intValue()).addDays(-1);
            /* Note: we are assuming that Subscription Term Unit is Day in the package
settings */
//          endDate = startDate.addDays(getEffectiveSubscriptionTerm(line).intValue() -
1);
        }
        return endDate;
    }

    private Decimal getEffectiveSubscriptionTerm(SObject quote, SObject line) {
        Decimal lineTerm = null;
        Date startDate =
(Date)line.get(String.valueOf(SBQQ__QuoteLine__c.SBQQ__EffectiveStartDate__c));
        Date endDate =
(Date)line.get(String.valueOf(SBQQ__QuoteLine__c.SBQQ__EffectiveEndDate__c));
        if ((startDate != null) && (endDate != null)) {
            /* Note: we are assuming that Subscription Term Unit is Month in the package
settings */
            lineTerm = startDate.monthsBetween(endDate.addDays(1));
            /* Note: we are assuming that Subscription Term Unit is Day in the package
settings */
//            lineTerm = startDate.daysBetween(endDate.addDays(1));
        } else {
            lineTerm =
(Decimal)line.get(String.valueOf(SBQQ__QuoteLine__c.SBQQ__SubscriptionTerm__c));
            if (lineTerm == null) {
                lineterm =
(Decimal)quote.get(String.valueOf(SBQQ__Quote__c.SBQQ__SubscriptionTerm__c));
                if (lineTerm == null) {
                    return
(Decimal)line.get(String.valueOf(SBQQ__QuoteLine__c.SBQQ__DefaultSubscriptionTerm__c));
                }
```

```
            }
        }
        return lineTerm;
    }

}
```

Javascript

```
export function onAfterCalculate(quote, lineModels) {
    var maxEffectiveEndDate = null;
    var maxEffectiveTerm = 0;
    if (lineModels != null) {
        lineModels.forEach(function (line) {
            var trueEndDate = calculateEndDate(quote, line);
            var trueTerm = getEffectiveSubscriptionTerm(quote, line);
            if (maxEffectiveEndDate == null || (maxEffectiveEndDate < trueEndDate)) {
                maxEffectiveEndDate = trueEndDate;
            }
            if (maxEffectiveTerm < trueTerm) {
                maxEffectiveTerm = trueTerm;
            }
            line.record["True_Effective_End_Date__c"] = toApexDate(trueEndDate);
            line.record["True_Effective_Term__c"] = trueTerm;
        });
        quote.record["True_Effective_End_Date__c"] = toApexDate(maxEffectiveEndDate);
        quote.record["True_Effective_Term__c"] = maxEffectiveTerm;
    }
}

function calculateEndDate(quote, line) {
    var sd = line.record["SBQQ__EffectiveStartDate__c"];
    var ed = line.record["SBQQ__EffectiveEndDate__c"];
    if (sd != null && ed == null) {
        ed = sd;
        ed.setUTCMonth(ed.getUTCMonth() + getEffectiveSubscriptionTerm(quote, line));
        ed.setUTCDate(d.getUTCDate() - 1);
    }
    return ed;
}

function getEffectiveSubscriptionTerm(quote, line) {
    var sd = line.record["SBQQ__EffectiveStartDate__c"];
    var ed = line.record["SBQQ__EffectiveEndDate__c"];
    if (sd != null && ed != null) {
        ed.setUTCDate(ed.getUTCDate() + 1);
        return monthsBetween(sd, ed);
    } else if (line.SubscriptionTerm__c != null) {
        return line.SubscriptionTerm__c;
    } else if (quote.SubscriptionTerm__c != null) {
        return quote.SubscriptionTerm__c;
    } else {
        return line.DefaultSubscriptionTerm__c;
```

```
    }
}

/**
 * Takes a JS Date object and turns it into a string of the type 'YYYY-MM-DD', which is
what Apex is expecting.
 * @param {Date} date The date to be stringified
 * @returns {string}
 */
function toApexDate(/*Date*/ date) {
    if (date == null) {
        return null;
    }
    // Get the ISO formatted date string.
    // This will be formatted: YYYY-MM-DDTHH:mm:ss.sssZ
    var dateIso = date.toISOString();

    // Replace everything after the T with an empty string
    return dateIso.replace(new RegExp('[Tt].*'), "");
}

function monthsBetween(/*Date*/ startDate, /*Date*/ endDate) {
    // If the start date is actually after the end date, reverse the arguments and multiply
 the result by -1
    if (startDate  > endDate) {
        return -1 * this.monthsBetween(endDate, startDate);
    }
    var result = 0;
    // Add the difference in years * 12
    result += ((endDate.getUTCFullYear() - startDate.getUTCFullYear()) * 12);
    // Add the difference in months. Note: If startDate was later in the year than endDate,
 this value will be
    // subtracted.
    result += (endDate.getUTCMonth() - startDate.getUTCMonth());
    return result;
}
```

Custom Package Total Calculation

Use a quote calculator plugin to calculate the total price for all components in a quote line and then store that value in the custom field Component Custom Total.

These are sample Apex and Javascript plugins for use in the JavaScript Quote Calculator. Each version exports all of the methods that the calculator looks for, and documents their parameters and return types.

EDITIONS

Available in: Salesforce CPQ Winter '16 and later

Javascript

```
export function onInit(lines) {
    if (lines != null) {
        lines.forEach(function (line) {
            line.record["Component_Custom_Total__c"] = 0;
        });
```

```
    }
};

export function onAfterCalculate(quoteModel, quoteLines) {
    if (quoteLines != null) {
        quoteLines.forEach(function (line) {
            var parent = line.parentItem;
            if (parent != null) {
                var pComponentCustomTotal = parent.record["Component_Custom_Total__c"] ||
 0;
                var cListPrice = line.ProratedListPrice__c || 0;
                var cQuantity = line.Quantity__c == null ? 1 : line.Quantity__c;
                var cPriorQuantity = line.PriorQuantity__c || 0;
                var cPricingMethod = line.PricingMethod__c == null ? "List" :
line.PricingMethod__c;
                var cDiscountScheduleType = line.DiscountScheduleType__c || '';
                var cRenewal = line.Renewal__c || false;
                var cExisting = line.Existing__c || false;
                var cSubscriptionPricing = line.SubscriptionPricing__c || '';

                var cTotalPrice = getTotal(cListPrice, cQuantity, cPriorQuantity,
cPricingMethod, cDiscountScheduleType, cRenewal, cExisting, cSubscriptionPricing,
cListPrice);
                pComponentCustomTotal += cTotalPrice;

                parent.record["Component_Custom_Total__c"] = pComponentCustomTotal;
            }
        });
    }
};

function getTotal(price, qty, priorQty, pMethod, dsType, isRen, isExist, subPricing,
listPrice) {
    if ((isRen === true) && (isExist === false) && (priorQty == null)) {
       // Personal note: In onAfterCalculate, we specifically make sure that priorQuantity
 can't be null.
        // So isn't this loop pointless?
        return 0;
    } else {
       return price * getEffectiveQuantity(qty, priorQty, pMethod, dsType, isRen, isExist,
 subPricing, listPrice);
    }
}

function getEffectiveQuantity(qty, priorQty, pMethod, dsType, isRen, exists, subPricing,
listPrice) {
    var delta = qty - priorQty;

    if (pMethod == 'Block' && delta == 0) {
        return 0;
    } else if (pMethod == 'Block') {
        return 1;
    } else if (dsType == 'Slab' && (delta == 0 || (qty == 0 && isRen == true))) {
        return 0;
```

```
    } else if (dsType == 'Slab') {
        return 1;
    } else if (exists == true && subPricing == '' && delta < 0) {
        return 0;
    } else if (exists == true && subPricing == 'Percent Of Total' && listPrice != 0 &&
delta >= 0) {
        return qty;
    } else if (exists == true) {
        return delta;
    } else {
        return qty;
    }
}
```

Apex

```
global class QCPWinter16Legacy implements SBQQ.QuoteCalculatorPlugin,
SBQQ.QuoteCalculatorPlugin2 {

    /* NOTE: the getReferencedFields method is no longer required if you use the
ReferencedFields field set on
 the Quote Line object.
             This field set must be created as it's not a managed one.
       NOTE: if you need to access Quote fields, you can create the ReferencedFields field
 set on the Quote object as well.
       NOTE: if you do not use the getReferencedFields method, you can remove
SBQQ.QuoteCalculatorPlugin2 from the class declaration.
    */
    global Set<String> getReferencedFields() {
        return new Set<String>{
            /* Note: add fields using the following format - Only add fields referenced

              by the plugin and not in the Line Editor field set on the Quote Line object

            String.valueOf(SBQQ__QuoteLine__c.My_Field_API_Name__c)
            */
            String.valueOf(SBQQ__QuoteLine__c.Component_Custom_Total__c),

            String.valueOf(SBQQ__QuoteLine__c.SBQQ__ProratedListPrice__c),
            String.valueOf(SBQQ__QuoteLine__c.SBQQ__PriorQuantity__c),
            String.valueOf(SBQQ__QuoteLine__c.SBQQ__PricingMethod__c),
            String.valueOf(SBQQ__QuoteLine__c.SBQQ__DiscountScheduleType__c),
            String.valueOf(SBQQ__QuoteLine__c.SBQQ__Renewal__c),
            String.valueOf(SBQQ__QuoteLine__c.SBQQ__Existing__c),
            String.valueOf(SBQQ__QuoteLine__c.SBQQ__SubscriptionPricing__c)
        };
    }

    global void onBeforePriceRules(SObject quote, SObject[] lines) {
    }

    global void onAfterPriceRules(SObject quote, SObject[] lines) {
    }
```

```
    global void onBeforeCalculate(SObject quote, SObject[] lines) {
    }

    global void onAfterCalculate(SObject quote, SObject[] lines) {
        for(SObject line : lines) {
            SObject parent =
line.getSObject(SBQQ__QuoteLine__c.SBQQ__RequiredBy__c.getDescribe().getRelationshipName());

            if(parent != null) {
              Decimal pComponentCustomTotal =
(Decimal)parent.get(String.valueOf(SBQQ__QuoteLine__c.Component_Custom_Total__c));

              Decimal cListPrice =
(Decimal)line.get(String.valueOf(SBQQ__QuoteLine__c.SBQQ__ProratedListPrice__c));
              Decimal cQuantity =
(Decimal)line.get(String.valueOf(SBQQ__QuoteLine__c.SBQQ__Quantity__c));
              Decimal cPriorQuantity =
(Decimal)line.get(String.valueOf(SBQQ__QuoteLine__c.SBQQ__PriorQuantity__c));
              String cPricingMethod =
(String)line.get(String.valueOf(SBQQ__QuoteLine__c.SBQQ__PricingMethod__c));
              String cDiscountScheduleType =
(String)line.get(String.valueOf(SBQQ__QuoteLine__c.SBQQ__DiscountScheduleType__c));
              Boolean cRenewal =
(Boolean)line.get(String.valueOf(SBQQ__QuoteLine__c.SBQQ__Renewal__c));
              Boolean cExisting =
(Boolean)line.get(String.valueOf(SBQQ__QuoteLine__c.SBQQ__Existing__c));
              String cSubscriptionPricing =
(String)line.get(String.valueOf(SBQQ__QuoteLine__c.SBQQ__SubscriptionPricing__c));

              pComponentCustomTotal = (pComponentCustomTotal == null) ? 0 :
pComponentCustomTotal;

              cListPrice = (cListPrice == null) ? 0 : cListPrice;
              cQuantity = (cQuantity == null) ? 1 : cQuantity;
              cPriorQuantity = (cPriorQuantity == null) ? 0 : cPriorQuantity;
              cPricingMethod = (cPricingMethod == null) ? 'List' : cPricingMethod;
              cDiscountSCheduleType = (cDiscountSCheduleType == null) ? '' :
cDiscountSCheduleType;
              cRenewal = (cRenewal == null) ? false : cRenewal;
              cExisting = (cExisting == null) ? false : cExisting;
              cSubscriptionPricing = (cSubscriptionPricing == null) ? '' :
cSubscriptionPricing;

              Decimal cTotalPrice = getTotal(cListPrice, cQuantity, cPriorQuantity,
cPricingMethod, cDiscountScheduleType, cRenewal, cExisting, cSubscriptionPricing,
cListPrice);
              pComponentCustomTotal += cTotalPrice;

              parent.put(SBQQ__QuoteLine__c.Component_Custom_Total__c,
pComponentCustomTotal);

            }
        }
```

```
    }

    global void onInit(SObject[] lines) {
        for(SObject line : lines) {
            line.put(SBQQ__QuoteLine__c.Component_Custom_Total__c, 0);
        }
    }

   private Decimal getTotal(Decimal price, Decimal quantity, Decimal priorQuantity, String
 pricingMethod, String discountScheduleType, Boolean renewal, Boolean existing, String
subscriptionPricing, Decimal ListPrice) {
        price = (price == null) ? 0 : price;
        renewal = (renewal == null) ? false : renewal;
        existing = (existing == null) ? false : existing;

        if(renewal == true && existing == false && priorQuantity == null) {
            return 0;
        } else {
            return price * getEffectiveQuantity(quantity, priorQuantity, pricingMethod,
discountScheduleType, renewal, existing, subscriptionPricing, listPrice);
        }

    }

    private Decimal getEffectiveQuantity(Decimal quantity, Decimal priorQuantity, String
pricingMethod, String discountScheduleType, Boolean renewal, Boolean existing, String
subscriptionPricing, Decimal ListPrice) {
        Decimal result = 0;
        Decimal deltaQuantity = 0;

        quantity = (quantity == null) ? 0 : quantity;
        priorQuantity = (priorQuantity == null) ? 0 : priorQuantity;
        pricingMethod = (pricingMethod == null) ? '' : pricingMethod;
       discountScheduleType = (discountScheduleType == null) ? '' : discountScheduleType;

        subscriptionPricing = (subscriptionPricing == null) ? '' : subscriptionPricing;
        renewal = (renewal == null) ? false : renewal;
        existing = (existing == null) ? false : existing;
        listPrice = (listPrice == null) ? 0 : listPrice;

        deltaQuantity = quantity - priorQuantity;

        if(pricingMethod == 'Block' && deltaQuantity == 0) {
            result = 0;
        } else {
            if(pricingMethod == 'Block') {
                result = 1;
            } else {
                if(discountScheduleType == 'Slab' && (deltaQuantity == 0 || (quantity ==
0 && renewal == true))) {
                    result = 0;
                } else {
                    if(discountScheduleType == 'Slab') {
                        result = 1;
```

```
                    } else {
                        if(existing == true && subscriptionPricing == '' && deltaQuantity
 < 0) {
                            result = 0;
                        } else {
                            if(existing == true && subscriptionPricing == 'Percent Of
Total' && listPrice != 0 && deltaQuantity >= 0) {
                                result = quantity;
                            } else {
                                if(existing == true) {
                                    result = deltaQuantity;
                                } else {
                                    result = quantity;
                                }
                            }
                        }
                    }
                }
            }

        return result;
    }

}
```

Find Lookup Records

Use a quote calculator plugin to query records within the plugin and use fields from those records to set each quote line's Description field.

These are sample Apex and Javascript plugins for use in the JavaScript Quote Calculator. Each version exports all of the methods that the calculator will look for, and documents their parameters and return types.

EDITIONS

Available in: Salesforce CPQ Winter '16 and later

Javascript

```
/**
 * Created by jfeingold on 9/27/16.
 */
export function onAfterCalculate(quote, lines, conn) {
 if (lines.length > 0) {
  var productCodes = [];
  lines.forEach(function(line) {
   if (line.record['SBQQ__ProductCode__c']) {
    productCodes.push(line.record['SBQQ__ProductCode__c']);
   }
  });
  if (productCodes.length) {
   var codeList = "('" + productCodes.join("', '") + "')";
   /*
    * conn.query() returns a Promise that resolves when the query completes.
    */
```

```
  return conn.query('SELECT Id, SBQQ__Category__c, SBQQ__Value__c FROM SBQQ__LookupData__c
 WHERE SBQQ__Category__C IN ' + codeList)
    .then(function(results) {
     /*
      * conn.query()'s Promise resolves to an object with three attributes:
      * - totalSize: an integer indicating how many records were returned
      * - done: a boolean indicating whether the query has completed
      * - records: a list of all records returned
      */
     if (results.totalSize) {
      var valuesByCategory = {};
      results.records.forEach(function(record) {
       valuesByCategory[record.SBQQ__Category__c] = record.SBQQ__Value__c;
      });
      lines.forEach(function(line) {
       if (line.record['SBQQ__ProductCode__c']) {
        line.record['SBQQ__Description__c'] =
valuesByCategory[line.record['SBQQ__ProductCode__c']] || '';
       }
      });
     }
    });
  }
 }
 return Promise.resolve();
}
```

Javascript - Method-Chaining

This plugin uses method-chaining style to construct the query, which is useful when you want to dynamically construct your queries.

```
/**
 * Created by jfeingold on 9/27/16.
 */
export function onAfterCalculate(quote, lines, conn) {
 if (lines.length) {
  var codes = [];
  lines.forEach(function(line) {
   var code = line.record['SBQQ__ProductCode__c'];
   if (code) {
    codes.push(code);
   }
  });
  if (codes.length) {
   var conditions = {
    SBQQ__Category__c: {$in: codes}
   };
   var fields = ['Id', 'Name', 'SBQQ__Category__c', 'SBQQ__Value__c'];
   /*
    * Queries can also be constructed in a method-chaining style.
    */
   return conn.sobject('SBQQ__LookupData__c')
    .find(conditions, fields)
    .execute(function(err, records) {
```

```
     if (err) {
      return Promise.reject(err);
     } else {
      var valuesByCategory = {};
      records.forEach(function(record) {
       valuesByCategory[record.SBQQ__Category__c] = record.SBQQ__Value__c;
      });
      lines.forEach(function(line) {
       if (line.record['SBQQ__ProductCode__c']) {
        line.record['SBQQ__Description__c'] =
valuesByCategory[line.record['SBQQ__ProductCode__c']] || '';
       }
      });
     }
    });
  }
 }
 return Promise.resolve();
}
```

Apex

```
global class QCPForFindingLookupRecords implements SBQQ.QuoteCalculatorPlugin,
SBQQ.QuoteCalculatorPlugin2 {
 global set<String> getReferencedFields() {
 return new Set<String> {
 String.valueOf(SBQQ__QuoteLine__c.SBQQ__ProductCode__c),
 String.valueOf(SBQQ__QuoteLine__c.SBQQ__Description__c)
 };
 }

 global void onInit(SObject[] lines) {}

 global void onBeforeCalculate(SObject quote, SObject[] lines) {}

 global void onBeforePriceRules(SObject quote, SObject[] lines) {}

 global void onAfterPriceRules(SObject quote, SObject[] lines) {}

 global void onAfterCalculate(SObject quote, SObject[] lines) {
 if (!lines.isEmpty()) {
 String[] productCodes = new String[0];
 for (SObject line : lines) {
 String productCode =
(String)line.get(String.valueOf(SBQQ__QuoteLine__c.SBQQ__ProductCode__c));
 if (productCode != null && !productCode.isWhitespace()) {
 productCodes.add(productCode);
 }
 }
 SBQQ__LookupData__c[] ds = [SELECT Id, SBQQ__Category__c, SBQQ__Value__c FROM
SBQQ__LookupData__c WHERE SBQQ__Category__c IN :productCodes];
 if (!ds.isEmpty()) {
 Map<String,String> valuesByCategory = new Map<String,String>();
```

```
 for (SBQQ__LookupData__c d : ds) {
 valuesByCategory.put(d.SBQQ__Category__c, d.SBQQ__Value__c);
 }
 for (SObject line : lines) {
 String productCode =
(String)line.get(String.valueOf(SBQQ__QuoteLine__c.SBQQ__ProductCode__c));
 if (productCode != null && !productCode.isWhitespace()) {
 line.put(String.valueOf(SBQQ__QuoteLine__c.SBQQ__Description__c),
valuesByCategory.get(productCode));
 }
 }
 }
 }
 }
}
```

Insert Records

Use JSForce in a javascript quote calculator plugin to insert records.

This is a sample Javascript plugin for use in the JavaScript Quote Calculator. It exports all of the methods that the calculator looks for, and documents their parameters and return types.

EDITIONS

Available in: Salesforce CPQ Winter '16 and later

```
/**
 * Created by jfeingold on 9/27/16.
 */
export function onAfterCalculate(quote, lines, conn) {
 if (lines.length) {
  var codes = [];
  lines.forEach(function(line) {
   var code = line.record['SBQQ__ProductCode__c'];
   if (code) {
    codes.push(code);
   }
  });
  if (codes.length) {
   var conditions = {
    SBQQ__Category__c: {$in: codes}
   };
   var fields = ['Id', 'Name', 'SBQQ__Category__c', 'SBQQ__Value__c'];
   return conn.sobject('SBQQ__LookupData__c')
    .find(conditions, fields)
    .execute(function(err, records) {
     console.log(records);
     if (err) {
      return Promise.reject(err);
     } else {
      var valuesByCategory = {};
      records.forEach(function(record) {
       valuesByCategory[record.SBQQ__Category__c] = record.SBQQ__Value__c;
      });
      var newRecords = [];
```

```
      lines.forEach(function(line) {
       var code = line.record['SBQQ__ProductCode__c'];
       var desc = line.record['SBQQ__Description__c'];
       if (code && desc && !valuesByCategory[code]) {
        newRecords.push({
         SBQQ__Category__c: code,
         SBQQ__Value__c: line.record['SBQQ__Description__c']
        });
       }
      });
      if (newRecords.length) {
       return conn.sobject('SBQQ__LookupData__c')
        .create(newRecords, function(err, ret) {
         console.log(ret);
        });
      }
     }
    });
  }
 }
 return Promise.resolve();
}
```

CPQ Product Search Plugin

Use the Product Search plugin to modify or add specificity to product filter searches and guided selling searches.

You can choose from two modes that control how the plugin modifies base Salesforce CPQ product searches.

- Enhanced: Add more parameters to the WHERE clause of the product search's existing SOQL query.
- Custom: Completely replace the product search's query logic with your own.

EDITIONS

Available in: All Salesforce CPQ Editions

Salesforce CPQ applies the modified search immediately upon entering the product search screen, so the initial group of searchable products is already filtered.

You can configure the Product Search plugin to filter a product search based on certain parameters when users enter their own search queries. For example, in Product Search, you could configure the plugin to return all search results in descending order from the most recent Last Ordered Date. When the user enters the Product Search, the products returned in the search results are shown from the most recent **Last Ordered Date**. Users can further filter through the Product Search filter panel, if necessary. We'll return to this example in the use case.

Plugin methods vary slightly between Product Search and Guided Selling. However, the different methods for each (isFilterHidden vs. inInputHidden & getFilterDefaultValue vs. getInputDefaultValue) effectively perform the same function.

Note: Date fields are returned as strings in the yyyy-mm-dd format.

Salesforce CPQ executes the implemented methods for Product Search in the following order.

```
* 1.0 Constructor()
 * 2.0 FOREACH(Search Input Field){
 *   2.1 isFilterHidden()
```

```
*   2.1 getFilterDefaultValue()
* }
* 3.0 isSearchCustom (CUSTOM vs ENHANCED)
* IF(isCustom){
*  4.0 search()
* }
* ELSE{
*   4.0 getAdditionalSearchFilters()
* }
```

Walkthrough

1. The Constructor method can be called first, but it's not required for implementation.
2. You'll start off by entering a search field value in the FOREACH method. Salesforce CPQ calls this method for each filter value it receives, then passes those values to the following methods:
   - isFilterHidden: Hides the search filter from the Quote Line Editor if the Quote status is set to approved
   - getFilterDefaultValue: Sets the field value for the initial search
3. Salesforce CPQ calls isSearchCustom to determine whether you're using Custom or Enhanced searching.
4. If isSearchCustom returned True, Salesforce CPQ calls search(). This method gives you full control of the search query - you'll build the Select Clause and Where Clause manually, then build and perform the query.
5. If isSearchCustom returned False, Salesforce CPQ calls getAdditionalSearchFilters. This method appends a WHERE clause to the existing SOQL query.

Method Samples

Each method below contains an implementation example. In each example, the SOQL query is from the price book entry table.**Constructor**

```
global class ExampleProductSearchPlugin implements SBQQ.ProductSearchPlugin{
 /**
  * Constructor. Not required for implementation
  */
 global ExampleProductSearchPlugin(){
  System.debug('METHOD CALLED: ExampleProductSearchPlugin Constructor');
 }
```

isFilterHidden

Description

Determines the visibility of a filter in the UI. Salesforce CPQ calls this implemented method for each input.

Parameters

Param	Type	Description
quote	SBQQ__Quote__c	Current quote object
fieldName	String	API Name of the Product2 Field

Return Values

Returns TRUE if the field should be hidden in the UI, FALSE otherwise.

Example

```
global Boolean isFilterHidden(SObject quote, String fieldName){
  /*
  // This would hide Product Code filter if Quote Status is Approved
  return fieldName == 'ProductCode' && quote.SBQQ__Status__c. == 'Approved';
  */
  return false;
 }
```

getFilterDefaultValue

Description

Determines the input value for the initial search. Salesforce CPQ calls this implemented method for each input field.

Parameters

Param	Type	Description
quote	SBQQ__Quote__c	Current quote object
fieldName	String	API Name of the Product2 field

Return Values

Value to pass for this field in the initial search. NULL if none.

Example

```
global String getFilterDefaultValue(SObject quote, String fieldName){
  System.debug('METHOD CALLED: getFilterDefaultValue');
  /*
  // This would set Product Family filter to Service if Quote Type is Quote
  return (fieldName == 'Family' && quote.SBQQ__Type__c. == 'Quote') ? 'Service' : NULL;
  */
  return NULL;
 }
```

isSearchCustom

Description

Determines if the mode is CUSTOM or ENHANCED. CUSTOM = Full control of Query; ENHANCED = Append additional criteria to WHERE clause.

Parameters

Param	Type	Description
quote	SBQQ__Quote__c	Current quote object
fieldValuesMap	Map<String,Object>	Map of search criteria. Key is Product2 API Name. Value is Value.

Return Values

TRUE for CUSTOM mode. FALSE for ENHANCED mode.

Example

```
global Boolean isSearchCustom(SObject quote, Map<String,Object> fieldValuesMap){
  System.debug('METHOD CALLED: isSearchCustom');
  /*
  // This would use CUSTOM mode if a Search field for sorting was defined and used
  return fieldValuesMap.get('Sort_By__c') != '';
  */
  return true;
 }
```

getAdditionalSearchFilters() [Called when isSearchCustom = FALSE]

Description

Appends an extra WHERE clause text when using ENHANCED mode. SOQL query is from the Price Book entry table.

Parameters

Param	Type	Description
Quote	SBQQ__Quote__c	Current Quote Object
fieldValuesMap	Map<String,Object>	Map of search criteria. Key is Product2 API Name. Value is Value.

Return Values

String to be appended to WHERE clause. NULL if none.

Example

```
global String getAdditionalSearchFilters(SObject quote, Map<String,Object> fieldValuesMap){

  System.debug('METHOD CALLED: getAdditionalSearchFilters');
  /*
  // This would add an extra inventory filter if the family is Hardware
  String additionalFilter = NULL;

  if(fieldValuesMap.get('Family') == 'Hardware'){
   additionalFilter = 'AND Product2.Inventory_Level__c > 3';
  }

  return additionalFilter;
  */
  return NULL;
 }
```

search() [Called when isSearchCustom = TRUE]

Definition

Override the entire search when using CUSTOM mode. Product2 fields in the Search Results field set should be set.

Parameters

Param	Type	Description

quote	SBQQ__Quote__c	Current Quote Object
fieldValuesMap	Map<String,Object>	Map of search criteria. Key is Product2 API Name. Value is Value. Only contains Keys for non-NULL values

Return Values

List of Price Book Entries with Product2 external lookup field set

Example

```
global List<Price BookEntry> search(SObject quote, Map<String,Object> fieldValuesMap){
  System.debug('METHOD CALLED: search');
  //GET ALL POSSIBLE FILTER FIELDS FROM THE SEARCH FILTER FIELD SET
  List<Schema.FieldSetMember> searchFilterFieldSetFields =
SObjectType.Product2.FieldSets.SBQQ__SearchFilters.getFields();

  //GET ALL POSSIBLE FIELDS FROM THE SEARCH RESULTS FIELD SET
  List<Schema.FieldSetMember> searchResultFieldSetFields =
SObjectType.Product2.FieldSets.SBQQ__SearchResults.getFields();

  //BUILD THE SELECT STRING
  String selectClause = 'SELECT ';

  for(Schema.FieldSetMember field : searchResultFieldSetFields){
   selectClause += 'Product2.' + field.getFieldPath() + ', ';
  }
        selectClause += 'Id, UnitPrice, Price Book2Id, Product2Id, Product2.Id';

        //BUILD THE WHERE CLAUSE
        String whereClause = '';

  for(Schema.FieldSetMember field : searchFilterFieldSetFields){
   if(!fieldValuesMap.containsKey(field.getFieldPath())){
    continue;
   }

   if(field.getType() == Schema.DisplayType.String || field.getType() ==
Schema.DisplayType.Picklist){
    whereClause += 'Product2.' + field.getFieldPath() + ' LIKE \'%' +
fieldValuesMap.get(field.getFieldPath()) + '%\' AND ';
   }
  }

  whereClause += 'Price Book2Id = \'' + quote.get('SBQQ__Price Book__c') + '\'';

  //BUILD THE QUERY
  String query = selectClause + ' FROM Price BookEntry WHERE ' + whereClause;

  //DO THE QUERY
  List<Price BookEntry> pbes = new List<Price BookEntry>();
  pbes = Database.query(query);
```

```
  return pbes;
 }
```

GUIDED SELLING

Salesforce CPQ executes the implemented methods for Guided Selling in the following order:

```
* 1.0 Constructor()
 * 2.0 FOREACH(Search Input Field){
 *   2.1 isInputHidden()
 *   2.1 getInputDefaultValue()
 * }
 * 3.0 isSearchCustom (CUSTOM vs ENHANCED)
 * IF(isCustom){
 *  4.0 search()
 * }
 * ELSE{
 *   4.0 getAdditionalSearchFilters()
 * }
```

Walkthrough

1. The **Constructor** method can be called first, but it's not required for implementation.
2. You'll start off by entering a search field value in the **FOREACH** method. Salesforce CPQ calls this method for each filter value it receives, then passes those values to the following methods:
3. **isInputHidden**: Hides the search filter from the Quote Line Editor if the Quote status is set to approved.
4. **getInputDefaultValue**: Sets the field value for the initial search.
5. Salesforce CPQ calls **isSearchCustom** to determine whether you're using Custom or Enhanced searching.
6. If isSearchCustom returned True, Salesforce CPQ calls **search()**. This method gives you full control of the search query - you'll build the Select Clause and Where Clause manually, then build and perform the query.
7. If isSearchCustom returned False, Salesforce CPQ calls **getAdditionalSearchFilters**. This method appends a WHERE clause to the existing SOQL query.

isInputHidden

Description

Determines the visibility of an Input in the UI when using Guided Selling. Salesforce CPQ calls this implemented method for each input.

Parameters

Param	Type	Description
quote	SBQQ__Quote__c	Current quote object
input	String	Name of the Quote Process Input

Return Values

Returns TRUE if the field should be hidden in the UI, FALSE otherwise

Example

```
global Boolean isInputHidden(SObject quote, String input){
  System.debug('METHOD CALLED: isInputHidden');
```

```
  /*
  // This would hide an Input called 'Urgent Shipment' on Fridays.
  return input == 'Urgent Shipment' && Datetime.now().format('F') == 5;
  */
  return false;
 }
```

getInputDefaultValue

Description

Determines the input value for the initial search. Salesforce CPQ calls this implemented method for each Quote Process Input.

Parameters

Param	Type	Description
quote	SBQQ__Quote__c	Current quote object
input	String	Name of the Quote Process Input

Return Values

Returns a value to pass for this field in the initial search. NULL if none.

Example

```
global String getInputDefaultValue(SObject quote, String input){
  System.debug('METHOD CALLED: getInputDefaultValue');

  return NULL;
 }
```

isSearchCustom()

Description

Determines if the mode is CUSTOM or ENHANCED when using Guided Selling. CUSTOM = Full control of Query; ENHANCED = Append additional criteria to WHERE clause.

Parameters

Param	Type	Description
quote	SBQQ__Quote__c	Current quote object
inputValuesMap	Map<String,Object>	Map of search criteria. Key is Process Input Name. Value is Value. Only contains keys for non-Null values. Includes additional key "qpid", for the Quote Process ID

Return Values

TRUE for CUSTOM mode. FALSE for ENHANCED mode.

```
/**
 global Boolean isSuggestCustom(SObject quote, Map<String,Object> inputValuesMap){
```

```
  return true;
 }
```

getAdditionalSearchFilters() [Called when isSearchCustom = FALSE]

Description

Appends an extra WHERE clause text when using Guided Selling in ENHANCED mode. SOQL query is from the Price Book entry table.

Parameters

Param	Type	Description
Quote	SBQQ__Quote__c	Current Quote Object
fieldValuesMap	Map<String,Object>	Map of search criteria. Key is Product2 API Name. Value is Value

Return Values

String to be appended to WHERE clause. NULL if none.

Example

```
global String getAdditionalSearchFilters(SObject quote, Map<String,Object> fieldValuesMap){

  System.debug('METHOD CALLED: getAdditionalSearchFilters');
  /*
  // This would add an extra inventory filter if the family is Hardware
  String additionalFilter = NULL;

  if(fieldValuesMap.get('Family') == 'Hardware'){
   additionalFilter = 'Product2.Inventory_Level__c > 3';
  }

  return additionalFilter;
  */
  return NULL;
 }
```

search() [Called when isSearchCustom = TRUE]

Definition

Override the entire search when using in CUSTOM mode. Product2 Fields in the Search Results field set should be set.

Parameters

Param	Type	Description
quote	SBQQ__Quote__c	Current Quote Object
fieldValuesMap	Map<String,Object>	Map of search criteria. Key is Product2 API Name. Value is Value. Only contains Keys for non-NULL values

Return Values

List of Price Book Entries with Product2 external lookup field set

Example

```
global List<Price BookEntry> search(SObject quote, Map<String,Object> fieldValuesMap){
  System.debug('METHOD CALLED: search');
  //GET ALL POSSIBLE FILTER FIELDS FROM THE SEARCH FILTER FIELD SET
  List<Schema.FieldSetMember> searchFilterFieldSetFields =
SObjectType.Product2.FieldSets.SBQQ__SearchFilters.getFields();

  //GET ALL POSSIBLE FIELDS FROM THE SEARCH RESULTS FIELD SET
  List<Schema.FieldSetMember> searchResultFieldSetFields =
SObjectType.Product2.FieldSets.SBQQ__SearchResults.getFields();

  //BUILD THE SELECT STRING
  String selectClause = 'SELECT ';

  for(Schema.FieldSetMember field : searchResultFieldSetFields){
   selectClause += 'Product2.' + field.getFieldPath() + ', ';
  }
        selectClause += 'Id, UnitPrice, Price Book2Id, Product2Id, Product2.Id';

        //BUILD THE WHERE CLAUSE
        String whereClause = '';

  for(Schema.FieldSetMember field : searchFilterFieldSetFields){
   if(!fieldValuesMap.containsKey(field.getFieldPath())){
    continue;
   }

   if(field.getType() == Schema.DisplayType.String || field.getType() ==
Schema.DisplayType.Picklist){
    whereClause += 'Product2.' + field.getFieldPath() + ' LIKE \'%' +
fieldValuesMap.get(field.getFieldPath()) + '%\' AND ';
   }
  }

  whereClause += 'Price Book2Id = \'' + quote.get('SBQQ__Price Book__c') + '\'';

  //BUILD THE QUERY
  String query = selectClause + ' FROM Price BookEntry WHERE ' + whereClause;

  //DO THE QUERY
  List<Price BookEntry> pbes = new List<Price BookEntry>();
  pbes = Database.query(query);

  return pbes;
 }
```

External Configurator Plugins

Developers can set up links to third-party web applications from within the configurator or quote line calculator. You can open links through custom actions or with an automatic launch through a pluggable configurator.

EDITIONS

Available in: Salesforce CPQ Winter '16 and later

Salesforce CPQ provides two ways to access an external configurator from the quote line editor.

- Create a custom action that launches a URL. When a sales rep clicks the custom action, Salesforce CPQ launches the external configurator.
- Define an external configurator in Salesforce CPQ package settings. Then, enable external configuration on any product that you want sales reps to use within that configurator. When a sales rep adds or configures that product in the quote line editor, Salesforce CPQ launches the external configurator. This feature is available as of Salesforce CPQ Summer '17.

Set Up an External Configurator to Launch from a Custom Action

Create a custom action that launches a non-Salesforce CPQ configurator.

Set Up a Web Application for Integration

Integrate with easyXDM to transfer data between your web application and Salesforce CPQ.

Set Up External Configurators for Automatic Launch

Allow users to launch a non-Salesforce CPQ configurator automatically from within the quote line editor. Salesforce CPQ captures the external configurator's values and applies relevant settings or changes back to the Salesforce CPQ configurator.

Set Up an External Configurator to Launch from a Custom Action

Create a custom action that launches a non-Salesforce CPQ configurator.

EDITIONS

Available in: Salesforce CPQ Winter '16 and later

You may have to add the following layouts and values.

- Add the Page and URL Target fields to the custom action page layout.
- Add the Popup value to the custom action's URL Target field.
- Add a label that represents your external configurator's name to the custom action's Label field.

1. From your Custom Actions tab, click **New**.
2. From the Label field, choose GIS.
3. From the Page field, choose Product Configurator.
4. From the URL Target field, choose Popup.
5. In the URL field, add a URL for a custom website that uses a secure https protocol.
6. Save your changes.

Important: We strongly recommend that you choose Dialog Window for the value of URL Target. Replace Window causes users to lose all their work when the external configurator loads.

Set Up a Web Application for Integration

Integrate with easyXDM to transfer data between your web application and Salesforce CPQ.

EDITIONS

Available in: Salesforce CPQ Winter '18 and later

1. Add the easyXDM library to your web application. You can find the library at http://easyxdm.net/wp/.
2. Use the easyXDM.RPC constructor with the following configuration. Set the configuration's keys to `postMessage`.

```
var rpc = new easyXDM.Rpc({},{
    remote: {
        postMessage: {} // This method is defined in SteelBrick
    },
    local: {
        postMessage: function(message) {
             // Do something with the JSON string that is received from SteelBrick.
        }
    }
});
```

3. After configuring the JSON, use Salesforce CPQ's postMessage method to send the JSON string back to the Salesforce CPQ app.

```
rpc.postMessage(JSON-string);
```

Set Up External Configurators for Automatic Launch

Allow users to launch a non-Salesforce CPQ configurator automatically from within the quote line editor. Salesforce CPQ captures the external configurator's values and applies relevant settings or changes back to the Salesforce CPQ configurator.

EDITIONS

Available in: Salesforce CPQ Winter '18 and later

1. From Setup, enter *Installed Packages*, and then select **Installed Packages**.
2. Find the Salesforce CPQ package and click **Configure**.
3. Select the **Additional Settings** tab.
4. In the External Configurator URL field, enter the URL for your external configurator.
5. Optional: On the Additional Settings page, you can also select the **Third Party Configurator** field. When active, Salesforce CPQ launches the external configurator so it takes up your entire screen. Any action that closes the close the external configurator (cancel, save, clicking **X** on your screen) redirects to the page that launched the external configurator.
6. Find the products that you want to launch the external configurator when a sales rep configures them.
   a. Select the **Externally Configurable** field on each product record.
   b. Make sure each product's Configuration Type field has a value of Required.

   The external configurator launches when a user clicks the wrench next to a configurable bundle, or when the user adds a product where a configuration event is required. If you set the configuration type to Allowed, the external configurator launches only when a sales rep selects the wrench icon next to a configurable bundle.

Page Security Plugin

Javascript page security plugins let developers control field-level visibility or data entry mode in Salesforce CPQ VisualForce pages.

EDITIONS

Available in: All Salesforce CPQ Editions

Let's review a few common page security plugin use cases.

- You want to show or hide an important field on each quote line. For example, you're selling training classes and you want to capture how many students are participating in the class. Set up your page security plugin so that the student number field shows only on quote lines related to your training class plugins.
- You want to make a field read-only or read-write based on its context. For example, you allow your users to specify the subscription term on each quote line, but you have some products that can only be quoted on a 12-month basis. A page security plugin can make the Subscription Term field read-only for such products, while keeping it read-write for the other products.

Example:

```
//PAGE SECURITY PLUG-IN START
export function isFieldVisible(fieldName, line) {

 if (fieldName == 'Security_Test__c') {
  if (line.Security_Level__c != 'Hidden') {
   return true;
  } else {
   return false;
  }
 }

 return null;

};

export function isFieldEditable(fieldName, line) {

 if (fieldName == 'Security_Test__c') {
  if (line.Security_Level__c == 'Read Only') {
   return false;
  }
 }

 return null;

};
//END PAGE SECURITY PLUGIN
```

Salesforce CPQ Electronic Signature Plugin

An electronic signature plugin lets developers add electronic signature functionality to their orgs. This is useful for organizations who wish to streamline processes involving signatures, such as finalizing purchases and contracts.

EDITIONS

Available in: All Salesforce CPQ Editions

Example:

```
global virtual interface ElectronicSignaturePlugin {
 void send(QuoteDocument__c[] documents);

 void updateStatus(QuoteDocument__c[] documents);

 void revoke(QuoteDocument__c[] documents);

 String getSendButtonLabel();
}


global interface ElectronicSignaturePlugin2 extends ElectronicSignaturePlugin {
 Boolean isSendButtonEnabled();
}
```

INDEX

A

Apex 28

C

Calculator plugin 3
CPQ 26–27
CPQ apex 28
CPQ plugin 1, 28
CPQ plugins 17

E

Electronic signature plugin 1, 28
External configurator 26–27

J

JSForce 16
JSQCP 2–4, 8, 13, 16

P

Page security plugin 1, 28
Plugin 1
Product search plugin 17

Q

QCP 2–4, 8, 13, 16
Quote calculator plugin 1
Quote Calculator Plugin 2, 4, 8, 13, 16

S

Salesforce calculator 3
Salesforce CPQ 1–4, 8, 13, 16, 26–28

W

Web application 26–27