**CPQ Plugins**

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

1. **Electronic Signature Plugin**: This will integrate Salesforce with electronic signature. plugins, which can be used to e-sign a quote document automatically. The e-signature tool will have instructions for how to fill in this detail
2. **Billing Plug-in:** Implements plug-ins for Salesforce Billing 1.0. No longer supported.
3. **Document Store Plug-in**: Document store plug-ins let you integrate Salesforce CPQ with document-management providers. Example DocuVault CloudFiles Drive Connect
4. **Recommended Products Plugin:** The Recommended Products plugin lets customers use their own product recommendation service in Salesforce CPQ, together with product search, search filters, favourites, and guided selling.

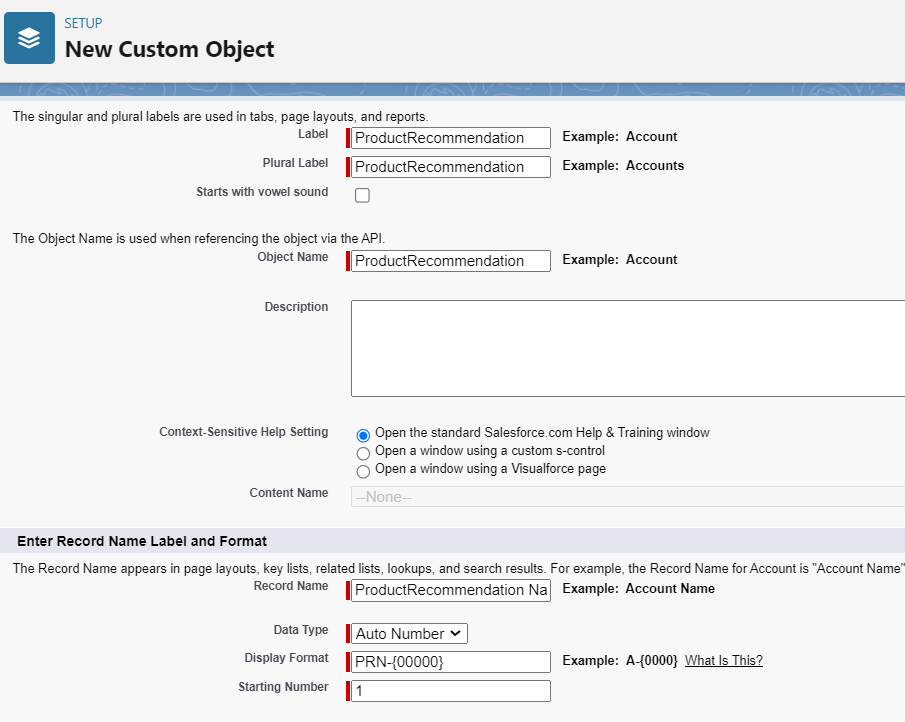
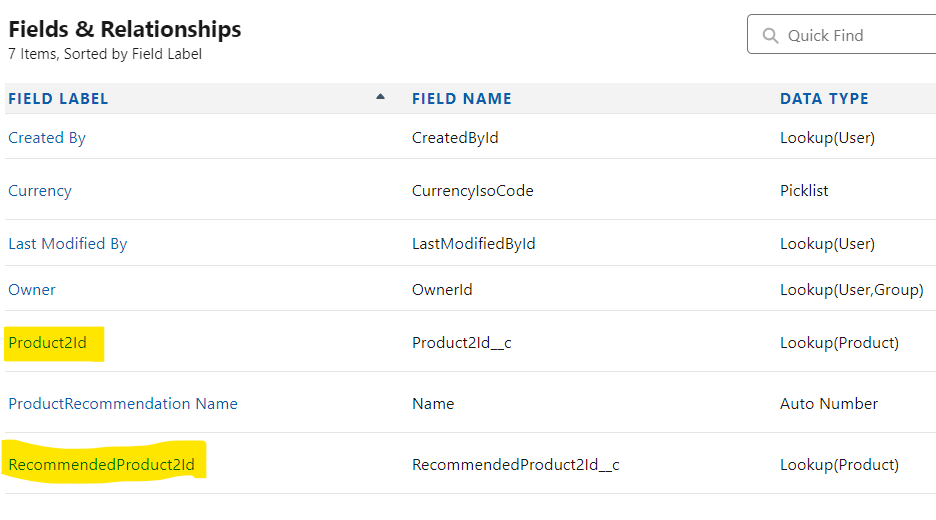
With the Recommended Products plugin, sales reps can see a list of recommended products while they are creating their quote. By making relevant products easier to find, the quoting process is more efficient, and sales reps can sell more products.   
  
Sample use cases of the Recommended Products plugin include:   
You can help sales reps close more deals by enabling them to create quotes with the ideal mix of products that anticipate their customers' needs.   
  
Sales reps can upsell more products by adding recommended products that are frequently sold with the products on their quotes.   
  
Complete these steps to implement the Recommended Product plugin-

On the Plugins tab in the Settings Editor, enable the Recommended Products Plugin

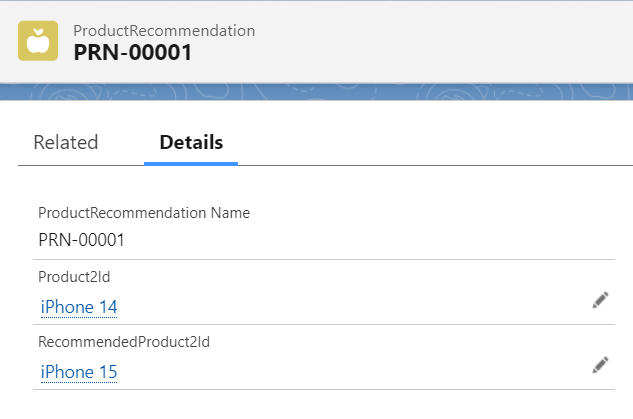
To add the Add Recommendations button to the quote line editor, activate the Add Recommendations custom action record. See Custom Actions.

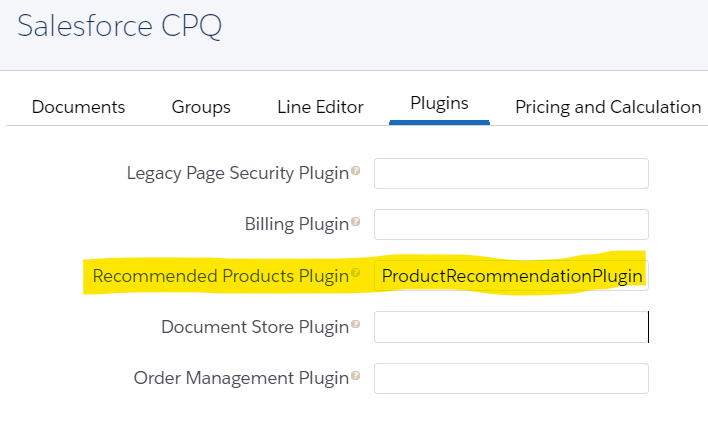
To fully enable the Recommended Products feature, you must provide your own implementation of the plugin interface. You can use your recommendation engine or a third-party service. Your plugin interface must implement the recommend() method in the global ProductRecommendationPlugin interface:

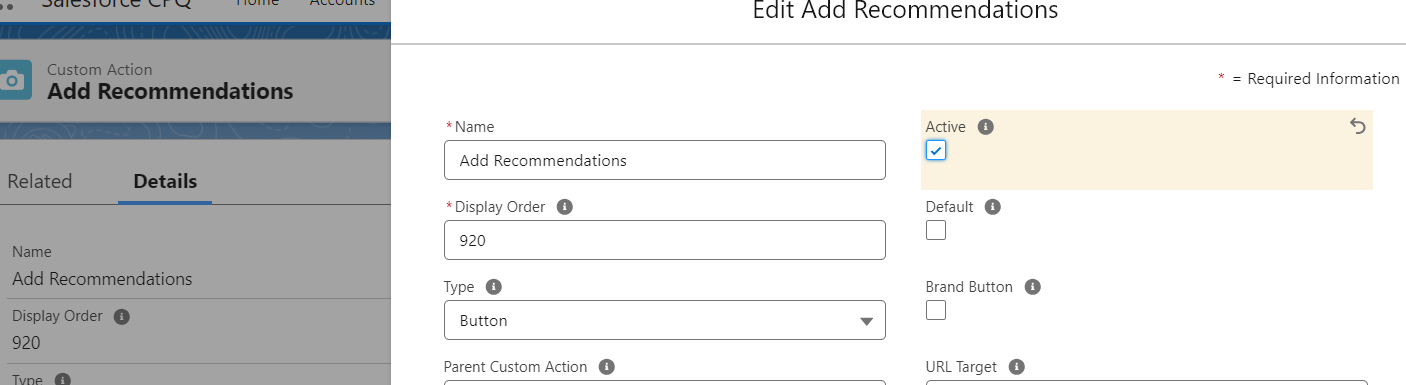
**Step 1:** Create a custom object name : ProductRecommendation

  
**Step 2:** Create new fields to the object – Product2Id (Lookup to Product), RecommendedProduct2Id (Lookup to Product)  


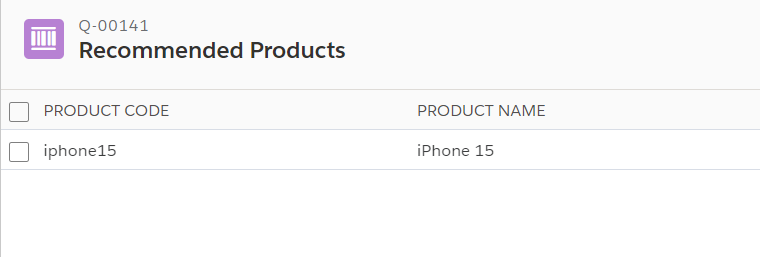
**Step 3**: Add a Tab for ProductRecommendation Object.

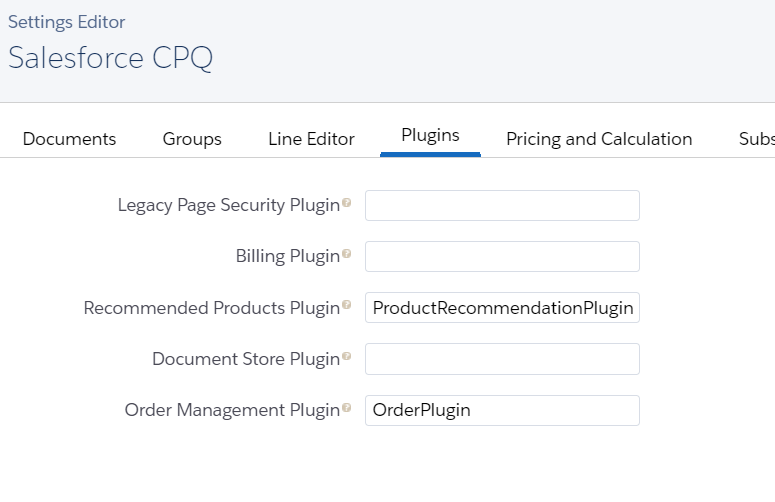
**Step 4:** Add iPhone 14 in Product2Id field and Add iPhone 15 in RecommendedProduct2ID field  
  
**Step 5:** Add an apex class – **ProductRecommendationPluginJH** ( Please [**check this**](https://developer.salesforce.com/docs/atlas.en-us.cpq_dev_api.meta/cpq_dev_api/cpq_recommended_products_plugin.htm) salesforce Link to check the code. Or [**Click here**](https://github.com/SFDCGYM/SalesforceCPQ/blob/main/ProductRecommendationPluginJH) to check it from Git.

**Step 6:** Add the name of the class to plugins from Package – Setup> Installed Package > Salesforce CPQ > Configure > Plugins Tab > Add **ProductRecommendationPluginJH .** Save the changes.  


**Step 7:** Enable Custom Action - Add Recommendations.  


**Step 8:** HGo to any Quote and add Iphone 14 and click on Add Recommendations drop down action. You will see iPhone 15 in the list.



1. **Order Management Plugin:**   
   An Order Management Plugin is used to set the Order's Start Date. A Quote ID is passed into the plugin and a Date is returned.   
     
   Below is an example plugin that delays the Order Start Date by the number of days specified in a field on the Quote. The plugin queries for the field value from the Quote, adds that number of days to the current system date, and returns the modified Order Start Date.   
     
   The name of the Apex class for the Order Management Plugin is entered under Setup > Installed Packages > Salesforce CPQ > Configure > Plugins > Order Management Plugin.   
     
   The 'Default Order Start Date' value (Setup > installed Packages > Salesforce CPQ > Configure > Order > Default Order Start Date) should be set to "-- None --", otherwise, it overrides the plugin.   
     
   **Step 1:** Create below Number Field on Quote Object - Delayed Start Days (Delayed\_Start\_Days\_\_c)  
     
   **Step 2**: Create an apex class name it **OrderPlugin**. [**Click here**](https://help.salesforce.com/s/articleView?id=000384532&type=1) to check it in Salesforce Page. Or [**Click here**](https://github.com/SFDCGYM/SalesforceCPQ/blob/main/OrderPlugin) to check it in Git page.   
     
   **Step 3:** Update the class name into the package setting. Go to Setup> Installed Package> Salesforce CPQ > Configure > Plugin > Add **OrderPlugin to** Order Management Plugin  
     
   **Step 4:** Go to any Quote and add 20 in Delayed Start Days and click on Ordered button. Open the order and you will notice that the start date is Today + Delayed Start Days(20)
2. **Legacy Page Security Plugin:** Salesforce CPQ has deprecated support for Apex page security plugins. And promoting to JavaScript Page Security Plugin. With the help of this plugin we can hide / unhide product fields dynamically to the users
3. **Legacy Quote Calculator Plugin:** Salesforce CPQ no longer provides support for Legacy Quote Calculator plugins. And promoting to use JavaScript Quote Calculator plugins.
4. **JavaScript Quote Calculator Plugin (QCP):** Add extra functionality to the quote line editor in Salesforce CPQ with custom JavaScript code. **Seven available methods** allow you to change how calculations are performed and manage page-level security such as field visibility.   
     
     
     
     
     
     
     
     
     
     
   When you develop a plugin for the JavaScript Quote Calculator, consider key guidelines   
   1. **QuoteModel and QuoteLineModel Type**:
      1. The JavaScript calculator represents Quote\_\_c and QuoteLine\_\_c objects QuoteModel and QuoteLineModel objects respectively.
      2. You can access the underlying sObjects 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 the 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”]
   2. **Using Promises:**
      1. A Promise is a build-in JavaScript object that allows for asynchronous programming in the browser Promises let you delay a certain action until another one has completed

**Scenarios where QCP can be used.**

* Complicated product-bundling guidelines.
* Conditional and complex pricing logic that's impossible to achieve through existing Salesforce

**CPQ pricing configurations.**

* Prevention from occlusion of the pricing engine, due to processing of large volumes of price rule or workflow data.

**QCP Required Configuration**

* Javascript Advanced Quote Calculator Plugin : Javascript resides in a Custom Script managed package object.
* Navigate to use the calculator plugin: Setup > Installed Packages > Configure "Salesforce CPQ" > Plugin tab > Quote Calculator Plugin: Name of custom script record

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();

};

1. **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();

};

1. **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();

};

1. **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();

};

1. **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();

};

1. **isFieldVisible – Page Security Plugin**

Note

This method can’t be used to alter data.

| **Param** | **Type** | **Description** |
| --- | --- | --- |
| {FieldName} | String | Name of the field that will be hidden or made visible |
| (QuoteLineModelRecord} | quoteLineModelRecord | Javascript representation of the SObject record of line you’re evaluating |

The calculator calls this method after it completes a calculation. Returns {Boolean}

export function isFieldVisible(fieldName, quoteLineModelRecord) {

if (fieldName == 'SBQQ\_\_Description\_\_c') {

return false;

}

return true;

};

1. **isFieldEditable - Page Security Plugin**

Note

This method can’t be used to alter data.

| **Param** | **Type** | **Description** |
| --- | --- | --- |
| {FieldName} | String | Name of the field that will be made read-only or editable |
| (QuoteLineModelRecord} | quoteLineModelRecord | Javascript representation of the SObject record of line you’re evaluating |

The calculator calls this method after it completes a calculation. Returns {Boolean}

export function isFieldEditable(fieldName, quoteLineModelRecord) {

if (fieldName == 'SBQQ\_\_Description\_\_c') {

return false;

}

return true;

};

* Example to use onInit Please [**Click Here**](https://github.com/SFDCGYM/SalesforceCPQ/blob/main/DisableProration)
* Example for isFieldEditable Please [**Click Here**](https://developer.salesforce.com/docs/atlas.en-us.cpq_dev_plugins.meta/cpq_dev_plugins/cpq_javascript_page_security_plugin.htm)

1. **Product Search Plugin:**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.   
     
   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. User can further filter through the Product Search filter panel, if necessary

Refer below link for different methods Available in the Product Search Plugin.   
  
<https://developer.salesforce.com/docs/atlas.en-us.cpq_dev_plugins.meta/cpq_dev_plugins/cpq_product_search_plugins.htm>

1. **Custom Action Plugin:**

Custom Action Plugin lets you run code before or after custom actions in salesforce cpq. Currently custom action plugins support only cloning actions.

| **Parameter** | **Type** | **Definition** |
| --- | --- | --- |
| quote | QuoteModel | A representation of the quote object. |
| clonedLines | Object | Properties:  clonedLines  Available with onAfterCloneLine. An array of new QuoteLineModels created from the clone action. When using onBeforeCloneLine, this property is undefined.  originalLines  Available with onBeforeCloneLine and onAfterCloneLine. An array of QuoteLineModels for the original quote lines that the user is cloning.  You can use the cloneLines parameter to change fields on the old and new quote lines. |
| conn | Object | A jsforce connection. |

**Example**: When we Clone a particular Quote Line, we want to check the Custom checkbox on cloned line.

**Step 1:** Create below checkbox on quote line object – Cloned\_Lines\_\_c

**Step 2:** Create script in Custom Script Object.[**Click Here**](https://github.com/SFDCGYM/SalesforceCPQ/blob/main/CloneScript) to check the code

**Interview Questions-**

1. I want to turn off proration for my subscription based products. How can I do that?
2. I want to extract a report on all quote lines that got cloned in QLE. How can I Do that?
3. Customer want to run product search on the bases of an external fields. How can I do that?
4. I want to disable a field from QLE if discount is greater than 40%. How can I do that?