TurboEngines

Table of Contents

Release Notes

Review the latest TurboEngines release notes that describe the system requirements and supported platforms, new features, enhancements, resolved issues, and known issues.

- Summer '21 Release Notes

Summer '21 Release Notes

These Release Notes contain the following information about Conga TurboEngines Summer '21 Release.

For more information on new features, enhancements, and document improvements, refer to

- What's New in *Conga TurboEngines Summer '21 Administrator Guide* or
- What's New in *Conga TurboEngines Summer '21 Data Sync Administrator Guide*

- Packages: Lists packages that are required to upgrade to this release of the product
- System Requirements and Supported Platforms: Lists requirements and recommendations for installing this release
- New Features: Provides high-level descriptions of new features introduced in this release, with links to more detailed information
- Enhancements: Provides high-level descriptions of enhancements to existing features
- Resolved Issues: Lists customer-reported issues that are resolved in this release or known issues resolved from previous releases
- Known Issues: Lists known issues that are applicable in this release

This documentation may contain descriptions of software features that are optional and for which you may not have purchased a license. As a result, your specific software solution and/or implementation may differ from those described in this document. Please contact your CSM or AE for information on your specific features and licensing.

Packages

The following packages and dependent packages are required to upgrade to this release to utilize all the new features of this release. These are the *minimum* required versions; later versions are also supported. Separate prerequisites for each feature can be found in the respective guides. The packages marked as **(New)** are new packages in this release.

Product	**Latest Certified Version** ***(Version Name | Version Number)***
Conga Base Library **(New)**	2.1.203 | 2.203
Conga Configuration & Pricing **(New)**	13.1921

Before installing or upgrading Conga managed packages, you must review the dependency matrix for each managed package. You can refer to the package dependency matrix at Conga Packages Dependency Matrix.

System Requirements and Supported Platforms

The following table lists the minimum requirements for installing and using Conga TurboEngines.

System Requirement	**Minimum Supported Version**
Operating System	Standard Salesforce.com requirements. See Salesforce PDF.
Browser	Conga supports the following browsers: • Microsoft Edge Chromium • Google Chrome Conga recommends the latest stable version of the browser for the best performance. Internet Explorer is not supported.

New Features

There are no new features in this release. Keep checking the Conga Documentation Portal for the latest updates.

Enhancements

The following section describes existing features that are changed (or are no longer supported) in this release.

Multi-tab feature in the TurboEngines Callbacks code editor

The user now has the ability to open multiple files of a callback project simultaneously in the code editor UI.

Get Started

For more information, see Managing TurboEngines Callbacks in *Conga TurboEngines Summer '21 Administrator Guide.*

Data Sync: Run History Admin User Tracking

An Admin User icon has been added to the Run History, and Run History Details pages to identify the admin user who triggered the corresponding data sync. You can click on the icon to display a pop-up with the admin user's username and email.

Get Started
For more information on tracking admin user activity from Run History, refer to the topic Working with Data Sync Run History in the *Conga TurboEngines Summer '21 Data Sync Administrator Guide.*

Data Sync: Ability to Resync based on user discretion

You can choose the list of objects when performing the resync for any object.

Get Started

For more information on resyncing, refer to Adding Objects and Fields for Sync *in the Conga TurboEngines Summer '21 Data Sync Administrator Guide.*

Data Sync: Run History Resync tracking

The Resync details are displayed separately on Run History UI to distinguish beside other sync types Manual and Scheduled syncs. In addition, this feature creates a separate log for the resync to track your changes to fields marked for sync.

Get Started

For more information on resyncing, refer to Adding Objects and Fields for Sync and Working with Data Sync Run History in the *Conga TurboEngines Summer '21 Data Sync Administrator Guide.*

Enhancement to Download Sync Error Report from Run Details

You can now download a collection (*.zip* file) of error messages in the format of .zip. The error messages report contains detailed error message information for debugging issues encountered during sync.

Get Started

For more information on downloading error message reports, refer to Viewing and Evaluating Error Messages in the *TurboEngines Summer '21 Data Sync Administrator Guide*.

Data Sync: Pricing Master Data Sync Details

The Run History Details page for objects synced to the Pricing Master Data (ElasticSearch) consumer endpoint now displays a link to provide additional details when one or more fields are rejected by the consumer endpoint (status in 'Partial Success' or 'Failed').

You can click the link to view the count of successful records, records that failed validation, and records that failed to sync.

Get Started

For more information on viewing sync details, refer to the topic Working with Data Sync Run History in the *Conga TurboEngines Summer '21 Data Sync Administrator Guide.*

Data Sync: Sync Index Sequence

When you create a sync index for any object from the Sync Settings UI, you can now choose the sequence of the fields you add to the index.

Get Started

For more information on creating and managing sync indexes, refer to Creating Sync Indexes in the *Conga TurboEngines Summer '21 Data Sync Administrator Guide.*

Data Sync: Identify Formula Field

The Manage Fields page in the Sync Settings UI now identifies formula fields to distinguish them from non-formula fields more easily.

Get Started

For more information on adding formula fields to the sync, refer to Adding Objects and Fields for Sync in the *Conga TurboEngines Summer '21 Data Sync Administrator Guide.*

Resolved Issues

There are no resolved issues in this release.

Known Issues

The following table provides the cumulative list of known issues up to this release.

Conga Internal ID	Description
CPQ-47814	In the TurboPricing flow, auto pricing occurs if you adjust any field on the cart while pricing in progress.
CMS-700	When using Match In Location, CPQ displays an error after adding the condition and action for different countries.
CMS-699	When you delete the condition of an Action, the action is also getting deleted along with the condition.
CMS-701	When configuring Match In Location, the error message is displayed continuously after deleting the actions.
CMS-702	In TurboConfig flow, CPQ displays an error after selecting the cloned condition.

Conga Internal ID	Description
CMS-703	When you update the quantity of cloned options after cloning it, the blank values are displayed for all the attributes of cloned options.
CMS-704	The original and cloned actions are deselected and added an extra clone for action after cloning a condition and action products.
CMS-705	Only one success message appears after adding the condition twice,
CMS-706	When performing repeat inclusion, an error message is not displayed after deleting the action from the mini cart.
CMS-707	When configuring Repeat Inclusion, the error message displayed for the prompt is disappeared.
CPQ-44694	The price factor (Bundle Only) adjustment is not supported.
CPQ-44665	Line item formula fields dependent on Salesforce default value fields should be added under view cart custom fields and custom pricing fields in config system properties.
CPQ-44599	Promotions with the "Contains" operator in the criteria do not work.
LS-4895	Some objects in seed data are not protected (made read-only) when the option "Include all fields including future fields" is enabled.

DOC ID: CTESUM21RN20210706

TurboEngines Documentation

Select one of the following for more information.

- About TurboEngines
- What's New in TurboEngines Documentation
- TurboEngines for Administrators

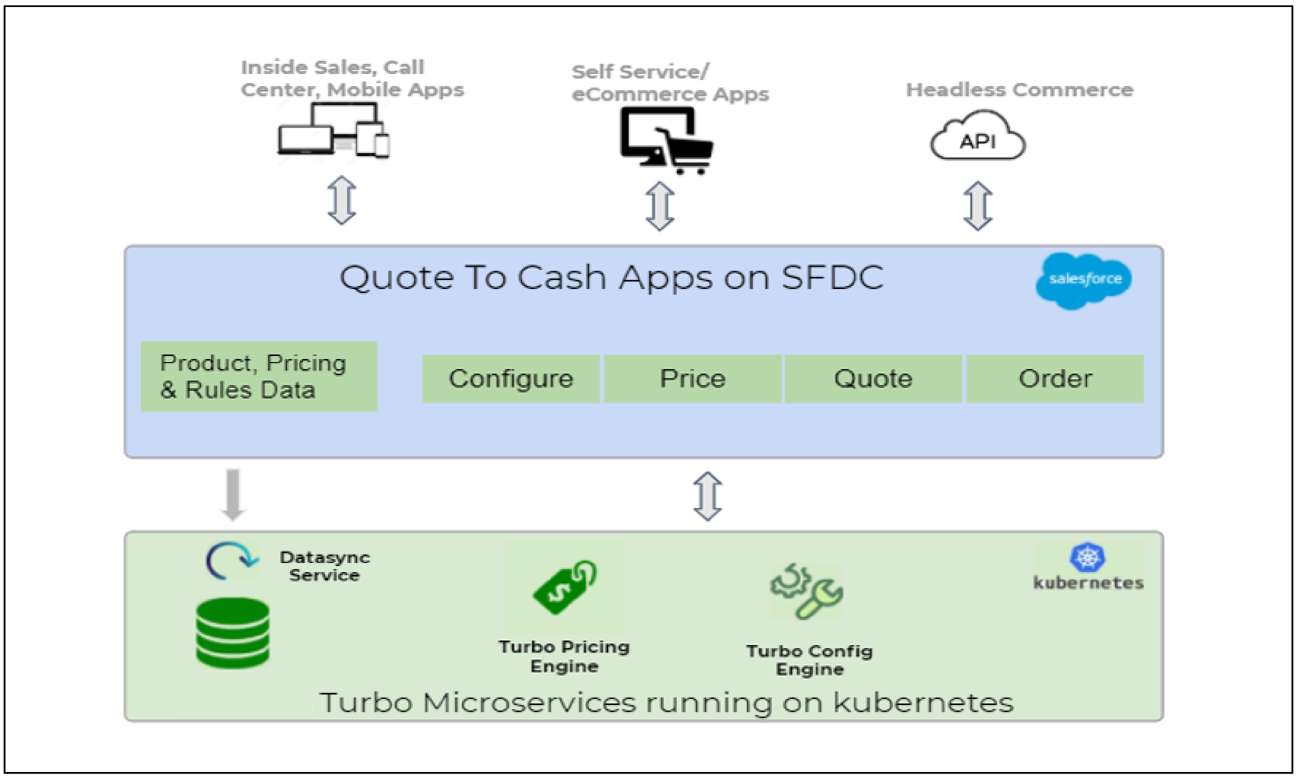
About TurboEngines

Conga TurboEngines is a concurrent processing engine provided by Conga comprising various microservices that process product configurations (TurboConfig), pricing calculations (TurboPricing), and other product-related business data, such as promotions. Conga TurboEngines offload the computation workload from the Salesforce platform to the Conga Flexible Compute Platform to reduce the processing time on the cart. In addition, processing the computation workload in the Conga Flexible Compute Platform reduces the interaction costs and the quote turnaround time, specifically during peak load or large transactions.

TurboEngines scale on the following dimensions:

- Number of users
- Size of transaction
- The complexity of the product and rules

TurboEngines also provide a critical component called TurboEngines Data Sync services that offer a high-performance mechanism to sync pricing and config master data at regular, scheduled intervals (or on-demand) between Salesforce and the Conga Flexible Compute Platform. Data is pushed to TurboPricing and TurboConfig consumer endpoints and made available for processing to take advantage of the performance improvements offered by the TurboEngines platform.


Inside Sales, Call Center, Mobile Apps
Self Service/ eCommerce Apps
Headless Commerce
API
Quote To Cash Apps on SFDC
salesforce
Product, Pricing & Rules Data
Configure
Price
Quote
Order
Datasync Service
Turbo Pricing Engine
Turbo Config Engine
kubernetes
Turbo Microservices running on kubernetes

About TurboConfig

TurboConfig is a configuration engine created to process product configuration rules when products and bundles are configured on a cart and when finalizing the quote. TurboConfig offloads the computation workload from the Salesforce platform to the Conga Flexible Compute Platform built using microservices to reduce the processing time of the configuration rules. Computation workload includes the processing of rules defined on the products. For example, in a TurboConfig enabled flow, when the Sales rep adds the product or the favorite configuration to the cart, the constraint rules associated with them are offloaded to the Conga Flexible Compute Platform to process. TurboConfig engine executes the rules, maintains rule states, and avoid unnecessary line item processing.

TurboConfig is recommended when you have a large number of rules or highly complex configuration rules to be applied while selecting a product or configuring a bundle.

To get started enabling TurboConfig for your org, refer to Enabling TurboEngines in an Org. To learn more about the TurboConfig service, refer to Frequently Asked Questions (TurboConfig).

Supported Features in TurboConfig

The following features and their capabilities are supported when TurboConfig mode is enabled. For information on the listed features, refer to *CPQ on Salesforce Administrator Guide.*

Feature	Capability	Supported
Constraint Rules	Inclusion rules	Yes
	Exclusion rules	Yes
	Validation rules	Yes
	Recommendation rules	Yes
	Replacement rules	Yes
	Product Scope: Product, Product Group, Product Family, Product Field Set	Yes
	Product Option Group scope	Yes It supports *condition* only.
	Match in Primary Lines or Options	Yes
	Match in Location	Yes
	Match in Asset	Yes
	Match in Cart Options	Yes
	Repeat Inclusion	Yes
	Condition Association	Yes
	Condition Criteria	Yes

Feature	Capability	Supported
	Action Criteria	Yes
	Match in Related Lines	No
	Is Bundle Context	Yes
Option Configuration	Min/Max Options	Yes
	Min/Max Total Quantity	Yes
	Is Hidden	Yes
	Is Picklist	Yes
	Modifiable Type	Yes
	Option Sequencing	Yes
	Default / Required Option	Yes
	Inclusion criteria	No
	Min/Max Quantity	Yes
	Quantity: Default, Modifiable	Yes
	Quantity: Auto Update	Yes
	Allow Cloning	Yes
	Config Type	No

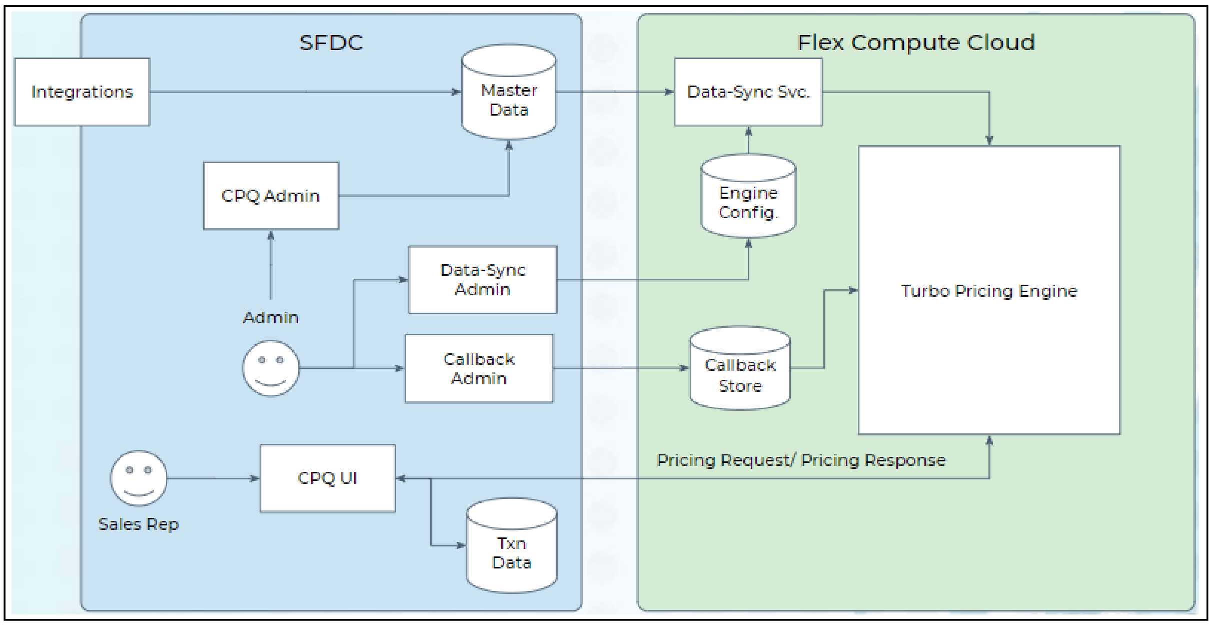
Feature	Capability	Supported
Product Attributes Display	Attribute: Read-Only, Hidden, Primary	Yes
	Two-column Attribute Display	Yes
	Three column Attribute Display	Yes
Product Attribute Rules	Product Scope: Product, Product Family, Product Group	Yes
	Filter Criteria	Yes
	Action Types: Allow, Default, Hidden, Disabled, Required, Reset	Yes
	Target field: Product Attribute Value, Line Item, Product, Pricelist, Product Configuration	Yes
Attribute Value Matrices	Product Scope: Product, Product Family, Product Group, Location	Yes
	Application Type: Default, Constraint, Force Set	Yes
Field Expressions / Rollup	Evaluation Context: Constraint Rule action, Record Update, Default Quantity, Rollup	Yes
	Update Product Attribute and Line Item object fields	Yes
	Rollup Group By Field: Line Item, Product Attribute	Yes
Callbacks	Option Filter Callback	No

Feature	Capability	Supported
Others	Formula field support (Condition/action)	Yes
	Lookups (For example, Attributes)	No
	TurboConfig Data Sync	Yes
	TurboPricing Integration	Yes
	ABO Flow support	Yes
	Multi-language support: For products & categories	Yes

About TurboPricing

TurboPricing is a pricing engine built using microservices to reduce the processing time on the cart. You can enable TurboPricing to offload complex pricing computation workload from the Salesforce platform to the Conga Flexible Compute Platform. It reduces time to submit prices to customers, improves user experience, and improves user adoption with a more responsive user interface.

The following diagram shows how data flows between a Salesforce org and Conga Flexible Compute Platform:



Supported Features in TurboPricing

The following table lists the features supported or not supported in TurboPricing. For information on the listed features, refer to *CPQ on Salesforce Administrator Guide.*

Feature	Sub-Feature	Available in TurboPricing
Price Waterfall	Manage Price Pipeline	Yes
	Manage Price Pipeline Ruleset	Yes
Price Calculation		
Bundle Pricing	Price included in bundle set at PLI	Yes
	Price included in bundle set at Bundle	Yes
	Price Adjustments at Bundle	Yes
	Rollup method Flat at Bundle level	Yes
	Rollup method Per Unit at Bundle level	Yes
	Auto sequencing of options	Yes. To use this feature, you must enable it in custom settings.
	Contract pricing when the same option exists in multiple bundles	No
Defaulting Quantity	Defaulting quantity for Bundles/ Multiple Charges	Yes
	Defaulting quantity FROM Product Attribute	Yes

Feature	Sub-Feature	Available in TurboPricing
	Default quantity derived FROM Advanced Formula	Yes
Defaulting Term	Defaulting term for Bundles/Multiple Charges	Yes
Price List Item		
Price Method	Use of Tiered Rates*	Yes
	Use of Per Unit Price method	Yes
	Use of Flat price Price method	Yes
Frequency	Use of Daily Frequency	No
	Use of Weekly Frequency	No
	Use of Frequencies Monthly, Quarterly, Half Yearly, Yearly	Yes
	Use of any Custom Frequency	No
Price Type	Use of Price Type - Included Usage	No
	Use of Price Type - One Time	Yes
	Use of Price Type - Per unit	Yes
	Use of Price Type - Usage	Yes
	Use of Price Type - Recurring	Yes
LineItem Update	Read-only quantity	Yes
	Read-only Selling term	Yes
Price Methods		

Feature	**Sub-Feature**	**Available in TurboPricing**
Proration	Allow Proration set on PLI	Yes
Price Method Per Unit	Use of Price Method Per Unit	Yes
Price Method Flat Price	Use of Price Method Flat price	Yes
Pricing Methods		
Min/Max Price	Min/Max Price applies to BasePrice	Yes
	Min/Max Price applies to BaseExtendedPrice	No
	Min/Max Price applies to ExtendedPrice	Yes
Price Ramps	Use of Price Ramps	No
	Use of Auto Ramp creation	No
	Use of Price ramp overlap	No
	Use of Price Escalators	No
	Use of Ramp Option without ramping bundle	No
Defer Pricing	Defer Pricing	No
Cost Models		
Cost Models	Cost Models	No
Conversions		

Feature	**Sub-Feature**	**Available in TurboPricing**
Currency Conversion	Use of Multi-Currencies	Yes
	Use of Dated Exchange Rates	Yes, it supports the following scenarios. • List Price • Price Matrices • Product Option Price • Conditional charge type • Custom field Pricing • Manual Adjustments • Related Pricing • Price Rule Sets (if and only if currency conversion is turned on) • Price Tiers • Formula Pricing (With Reference Type) • Rounding mode • Promotion • Proration • Contract Price
	Disable Currency Conversion Rate for a Price List	Yes

Feature	**Sub-Feature**	**Available in TurboPricing**
	Currency conversion enabled at a rule see the level	Yes
UOM Conversion	Use of UOM Conversions	Yes
	Use of Product specific conversion rates	Yes
	Use of Product Family-specific conversion rates	Yes
Price Method		
Price Method	Use of Price Method Percentage	No. You must switch to Related Price List Items.
Related Pricing		
Related Pricing	Use of Related Price Lists in Pricing	No. You must switch to Related Price List Items.
	Source PLI set on Related Price List Item	Yes
	Source product and charge typeset on Related Price List Item	Yes
	Source product group and charge typeset on Related Price List Item	Yes
	Source product family and charge typeset on Related Price List Item	Yes
	Source Custom Group and charge typeset on Related Price List Item	Yes

Feature	Sub-Feature	Available in TurboPricing
Adjustments in Related Pricing	Adjustment defined on PLI	Yes
	Adjustment defined on Related Price List Item	Yes
Price Rule Set		
Header Scope and Criteria	Use of Effectivity period and Active flag	Yes
	Use of Scope Fields - price list, charge type, product family, product category, product group	Yes
	Use of Advanced Criteria	Yes
	Use of Advanced Criteria with Line Item Reference Fields	Yes
	Use of Wildcards in Advanced Criteria	Yes
	Application Level Bundle or Line Item	Yes
	Application Level Aggregate	No
	Use of StopProcessingMoreRules flag	Yes
Dimension based Price Rules	Use of StopProcessingMoreRules flag	Yes
	Use of Adjustment applies to - Base Price	Yes
	Use of Adjustment applies to - Base Extended Price	Yes
	Use of Adjustment applies to - Extended Price	No

Feature	Sub-Feature	Available in TurboPricing
Criteria based Price Rules	Use of StopProcessingMoreRules flag	Yes
	Use of Adjustment applies to - Base Price	Yes
	Use of Adjustment applies to - Base Extended Price	Yes
	Use of Adjustment applies to - Extended Price	No
	Match in Product Group	Yes
	Match in Asset	No
Price Dimension		
Use of Un-supported Price Dimension Types	Use of Un-supported Price Dimension Types - any type Except Line Item, Product Attribute, and Formula Field	No
Use of Custom Price Dimension Types	Use of Custom Price Dimension Types	No. You must convert this to Formula Field.
Service CPQ	Service CPQ	No
Adjustments		
Manual Adjustments	Line level adjustments	Yes
	Group adjustments	Yes
	Group adjustment spread	Yes
	Line-level adjustment of usage price tiers	Yes

Feature	Sub-Feature	Available in TurboPricing
	Usage Tier Modifiable	Yes
	Misc Charge Types	Yes
Adjustments	Adjustment Bucketing	No
	Ability to create multiple Adjustments	No
	Auto-refresh Usage price tiers	No
	Calculate Net Adjustment % before rounding the Net Price	No
Bundle/Option level manual Adjustments	Line-level adjustments	Yes
Bundle/Option level Adjustments	Group adjustments	Yes
Asset-Based Pricing		
Assets	Assets	Yes, Partially available in TurboPricing.
Incentives		
Other application types except Promotion	Use of Price Program, Loyalty, Rebate, Milestone Incentive or Custom	No
Promotions applied on Line Item and Summary Group	Promotions applied on Line Item and Summary Group	Yes
Promotions applied on other items	Promotions applied on other items except Line Item and Summary Group	No
Support for Promotion type - Own every X Get Y, Support for Promotion type - Own every X Get Y	Support for Promotion type - Own every X Get Y	No

Feature	**Sub-Feature**	**Available in TurboPricing**
Support for other Promotion types - except Own every X Get Y, Support for Promotion types - except Own every X Get Y	Support for other Promotion types - except Own every X Get Y	Yes
Incentive Limits	Support for Promotion Limits	Yes
Incentive Coupons	Support for Coupon Limits	No
Sales Promotions	Sales Promotions	No
Advanced Criteria set in Price Ruleset	Support for Incentive Criteria on Price Ruleset	Yes
Advanced Criteria with Reference Fields on Price Ruleset	Advanced Criteria with Reference Fields on Price Ruleset	Yes
Quotes		
Quote collaboration	Quote collaboration	No
Carts		
Favorite Configurations	Use of Favorite Configurations	Yes
Smart Carts	Use of Smart Carts	No
Submit for Approval	Submit for Approval	Yes
Copy	Copy products	Yes
Cart Line Item		
Revalidate	Revalidate**	Yes, Partially supported in TurboPricing.

Feature	**Sub-Feature**	**Available in TurboPricing**
Totaling and Summary Groups		
Adhoc Totaling	Adhoc Totaling	Yes
Deal Guidance		
Deal Guidance	Deal Guidance	Yes
Callbacks		
Callbacks	Pricing Callback	Yes
	Validation Callback	Yes
	Cart Approval Callback	Yes
	Advanced Approval Callback	No
	Loyalty Cycle Callback	No
	Bulk Loyalty Point Callback Class	No
	Adjustment Spread Callback	Yes. You can use the new TurboPricing Callback instead.
	Loyalty Point Callback	No
	Related Pricing Callback	Yes
	Pricing Extension Callback	Yes. You can use the new TurboPricing Callback instead.
User Experience		

Feature	Sub-Feature	Available in TurboPricing
Bundle-specific option line item update	Read-only quantity	Yes
	Read-only selling term	Yes
LineItem Update	Line level adjustments	Yes
	Group adjustments	Yes

Installing TurboPricing Assessment Package

Apttus_TPAssessment package contains the TurboPricing Assessment page and related resources. Installing this package creates a TurboPricing Assessment tab. This tab lists the list of features that are currently used and supported features in TurboPricing.

This package depends on the CPQ package (Spring '19 or later).

Pre-requisites

You must rename the instance names for Config System Properties and Config Custom Classes as mentioned below.

Custom Setting Name	Instance Name
Config System Properties	System Properties
Config Custome Classes	Custom Classes

To install TP_Assessment Package

What's New in TurboEngines Documentation

The following table lists changes in documentation to support each release.

Summer '21

Document	Publication Date	Topic	Description
Summer '21	06 Jul 2021	Managing TurboEngines Callbacks	Updated topic. Added a note about the multi-tab feature.
		PricingHelper	Updated topic. Added new methods.
		Pricing Base Price Callback Interface	Updated topic. Added new extension points.
		Guidelines for TurboEngines Callbacks Coding	New topic.
		Migrating Salesforce Pricing Callback to TurboEngines Pricing Callback	New topic.

Spring '21

Document	Topic	Description
Spring '21 REV B	Supported features in TurboPricing	Updated topic. Updated the table with the newly supported feature.
Spring '21 REV A	Navigating the TurboEngines Callbacks Administrator User Interface	Updated topic. Updated the content and screens as per the new UI enhancements.
	Managing TurboEngines Callbacks	Updated topic. Updated the content and screens as per the new UI enhancements.

Document	Topic	Description
Spring '21	Supported features in TurboPricing	Updated topic. Updated the table with newly supported features.
	Supported Features in TurboConfig	Updated topic. Updated the table with newly supported features.
	Preparing Tenant Information	Updated topic.
	Version History	New Topic.
	Validate QueryModel	New Topic
	Downloading a Callback	New Topic.
	Pricing Callback Class for TurboPricing	Updated Topic. Updated the description.
	Pricing Base Price Callback Interface	Updated Topic. Added a note about the code snippet usability.
	Pricing Totalling Callback Interface	Updated Topic. Added a note about the code snippet usability.
	Related Pricing Callback Interface	Updated Topic. Added a note about the code snippet usability.
	Managing TurboEngines Callbacks	Updated Topic. Added the table with new features.

Winter '20

Document	Topic	Description
Winter 2020	Supported features in TurboPricing	Updated topic. Updated the table with newly supported features.
	Supported features in TurboConfig	Updated topic. Updated the table with newly supported features.

Document	Topic	Description
	DB Helper	Updated topic.
	CacheHelper	Updated topic.
	VaultHelper	Updated topic.
	Onboarding Data Sync Services	Updated topic.
	Pricing Base Price Callback Interface	Updated topic.
	Pricing Totalling Callback Interface	Updated topic.
	Related Pricing Callback Interface	New topic.
	Importing a Callback	New Topic. This feature provides a mechanism to import the callback project.
	Helper Functions for TurboEngines Callbacks	Renamed the topic from "Helper Functions for TurboPricing Callbacks".
	Configuring TurboEngines Callbacks	Renamed the topic from "configuring TurboPricing Callbacks".
	Navigating the TurboEngines Callbacks Administrator User Interface	Renamed the topic from "Navigating the TurboPricing Callbacks Administrator User Interface".
	Managing TurboEngines Callbacks	Renamed the topic from "Managing TurboPricing Callbacks".
	Syncing TurboConfig Data	Deleted Topic.

Summer '20

Document	Topic	Description
Summer 2020 Rev B	Validation Callback Class	Deleted Topic.
	Supported features in Apttus TurboPricing	Updated topic. Updated the table with newly supported features.
Summer 2020 Rev A	Configuring Data Sync for TurboPricing	Moved topic to "Onboarding data sync services".
	Navigating the TurboEngines Callbacks Administrator User Interface	Replaced screenshots.
	Managing TurboEngines Callbacks	Replaced screenshots.
	Helper Functions	Renamed the topic to "Helper Functions for TurboPricing Callbacks" and moved the new topic under "Configuring TurboPricing Callbacks".
	Helper Functions for TurboEngines Callbacks	Renamed from "Helper Functions" and moved out of "Pricing Callback Class for TurboPricing".
	CacheHelper	New topic, moved out of "Helper Functions for TurboPricing Callbacks".
	DBHelper	New topic, moved out of "Helper Functions for TurboPricing Callbacks".
	HttpHelper	New topic, moved out of "Helper Functions for TurboPricing Callbacks".

Document	Topic	Description
	LogHelper	New topic, moved out of "Helper Functions for TurboPricing Callbacks".
	MetadataHelper	New topic, moved out of "Helper Functions for TurboPricing Callbacks".
	PricingHelper	New topic, moved out of "Helper Functions for TurboPricing Callbacks".
	VaultHelper	New topic, moved out of "Helper Functions for TurboPricing Callbacks".
Summer 2020	All topics	First release

TurboEngines for Administrators

Conga TurboEngines Administrator Guide provides information to configure TurboEngines: TurboConfig and TurboPricing. Application administrators and Conga customer administrators can also use content in this guide to perform updates to configurations, configure settings, and other microservice functionalities.

Topic	Description
What's Covered	This guide walks the administrator through the process of TurboEngines administration. It provides conceptual information, step-by-step instructions to deploy and configure Conga TurboEngines for integrated systems.
Primary Audience	• TurboEngines Implementation Teams • Customer Administrators
IT Environment	Refer to the latest *Conga TurboEngines Release Notes* for information on System Requirements and Supported Platforms.
Other Resources	• *Conga TurboEngines Data Sync Documentation* • *Conga CPQ Documentation*

This guide describes the following tasks:

- Reviewing the list of supported features for TurboConfig and TurboPricing
- Configuring Conga TurboEngines
- Enabling TurboEngines
- Completing Pre-Provisioning Tasks
    - Creating a Connected App
    - Preparing Tenant Information
- Completing Post-Provisioning Tasks for TurboConfig
    - Configuring Remote Site Settings
    - Configuring Custom Settings
    - Configuring Custom Flows
    - Syncing TurboConfig Product Data
- Completing Post-Provisioning Tasks for TurboPricing
    - Configuring TurboPricing Settings
    - Customizing TurboPricing Callbacks
    - Configuring data sync settings
    - Syncing TurboPricing Pricing Data

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

- Basic Salesforce administration
- Salesforce Lightning experience
- Salesforce and Conga terms and definitions
- Conga TurboPricing Overview and Data Sync architecture
- Basic understanding of Conga TurboPricing and TurboConfig

DOC ID: CTESUM21AG20210706

Select one of the following topics for more information:

- Configuring Conga TurboEngines
- Frequently Asked Questions (TurboConfig)

Configuring Conga TurboEngines

The topics in this section provide information and step-by-step tasks for enabling TurboEngines for your organization.

ⓘ Please start with the Enabling TurboEngines in an Org topic and refer to the necessary steps you must take before and after the TurboEngines provisioning process.

- Enabling TurboEngines in an Org
- Creating a Connected App
- Preparing Tenant Information
- Post-Provisioning Tasks (TurboConfig)
- Post-Provisioning Tasks (TurboPricing)
- Onboarding Data Sync Services

Enabling TurboEngines in an Org

This topic provides a summary of the necessary steps for enabling TurboEngines (TurboConfig and TurboPricing) for your org.

An administrator can be any of the following persona: Customer Administrator, Partner Administrator, any other administrators assigned the responsibility of enabling TurboEngines for their org. In the table on this topic, this persona is referred to as the *Tenant Admin*.

ⓘ **You cannot use TurboConfig and TurboPricing simultaneously.**

Prerequisites

- Check the "Supported Features" topics (under About TurboEngines) for the service you want to enable. Make sure all of the features you want are included before making a provisioning request.
- You must have the appropriate TurboEngines license before turning on your org. If you do not have a license, please reach out to your Conga Account Executive.
- You must have the Summer 2020 or later build of Conga Configuration & Pricing (Conga CPQ) in the Salesforce org to enable TurboConfig and TurboPricing. Refer to "Packages" in Conga *CPQ Summer '21 Release Notes*.

Enabling TurboEngines

To enable TurboEngines, perform the following steps for each org:

Step	Task	Owner	Description
Pre-Provisioning Tasks			
1	Set up Connected App in your org	Tenant Admin	Create a connected app to provide authentication and authorization to TurboConfig and TurboPricing Data Sync Service.
2	Prepare pre-provisioning tenant information	Tenant Admin	Gather all required information for provisioning your TurboConfig or TurboPricing org. Provide this information to Conga Technical Support to begin the provisioning process.
Post-Provisioning Tasks Perform the following steps only after receiving a notice from Conga Technical Support that the requested orgs are provisioned. You must have the new service URLs to proceed.			
3	Set up Remote Site Settings (TurboConfig)	Tenant Admin	Use the service URL you received from Conga Technical Support to set up the remote site settings for TurboConfig.

Step	Task	Owner	Description
4	Configure Services	Tenant Admin	For TurboConfig, update the following settings: 1. Set up the Config Execution Mode 2. Set up the custom flow and Configure Products button For TurboPricing, update the following settings: 1. Set up the Pricing Execution Mode 2. Set up the TurboPricing endpoint URL
5	Configure TurboEngines Callbacks	Tenant Admin	Configure TurboEngines Callbacks. • TurboPricing Callbacks
6	Configure data sync service	Tenant Admin	Configure specific settings to onboard data sync services.
7	Sync data to TurboEngines	TurboEngines Administrator (can be Tenant Admin)	Set up and schedule or activate data sync to sync master data.

Creating a Connected App

As part of the pre-provisioning process you must configure a Connected App in your org to provide authentication and authorization for the following TurboEngines services:

- TurboConfig
- TurboPricing

> ⓘ The example in the following tasks is provided for TurboConfig but is the same process for any service configuration.

To create a Connected App

1. Navigate to **Setup** > **App Setup** > **Create** > **Apps.**
2. Scroll down and search for the **Connected Apps** related list and click **New** to create a new app.
3. Fill in the following details in the **Basic Information** section.

Field	Description
Connected App Name	Enter the name of the Connect App.
API Name	The API name is generated automatically based on the name of the Connected App.
Contact Email	Enter the email address of the administrator managing the Connected App.

4. Fill in the following details in the **API (Enable OAuth Settings)** section.

Fields	Description
Enable OAuth Settings	Select this to define the OAuth settings. For example, *TurboConfig*. When you enable this field, additional settings are displayed under **API (Enable OAuth Settings)** section.
Enable for Device Flow	Select this to enable the connected app for an external application.

Fields	Description
Callback URL	The **Callback URL** is generated automatically when you select the field **Enable for Device Flow**. For example, *https://test.salesforce.com/services/oauth2/success* is generated based on the instance URL. You can also add other URLs in separate lines.
Selected OAuth Scope	Select all the entries under **Available OAuth Scopes** and move them to **Selected OAuth Scopes** by clicking the **Add** arrow.
Require Secret for Web Server Flow	Select this to require the connected app to provide a consumer secret for authorization.

5. You must leave all other fields blank. Click **Save**.

To capture Consumer Key and Consumer Secret

After you create a Connected App, CPQ generates **Consumer Key** and **Consumer Secret**. You must provide the values of **Consumer Key** and **Consumer Secret** to Conga Technical Support.

1. Navigate to **Setup** > **App Setup** > **Create** > **Apps.**
2. Scroll down and search for the **Connected Apps** related list.
3. Click the name of the Connected App you created in the previous topic.
4. Click **Copy** next to **Consumer Key**.
5. Click **Click to reveal** next to **Consumer Secret**. After the value of the field is displayed, click **Copy**.
6. Store the information for the next part of the process.

Preparing Tenant Information

Your provisioning request for TurboConfig or TurboPricing must include specific information related to your tenant. Before your org can be provisioned, you must gather the required information and provide it to Conga Technical Support. What information must be collected will differ depending on the service you are provisioning.

ⓘ Configure a Connected App to use with TurboEngines before collecting the information described in this topic.

Refer to the following table for all required pre-provisioning information:

While implementing TurboPricing, any Id (15-character Id) that is returned from Salesforce is converted to an 18-character Id for the proper functioning of TurboPricing.

Configuration	Required for Service	Description
OrgId	TurboConfig, TurboPricing	This is the Salesforce Organization ID of the org to be provisioned for the TurboEngine service. To locate your Organization ID: 1. Log in to the org to be provisioned. 2. Go to **Setup** > **Company Profile** > **Company Information** > **Salesforce.com Organization ID**. 3. Copy the 15-character ID (to be converted into 18 characters). You can add any random characters to the Org ID for conversion. For example, TenantId = *00d3i000000qn7xAAA*
Tier	TurboConfig, TurboPricing	The Tier (Gold, Silver, and Bronze) to be provisioned. If no tier is provided, then **Bronze** is selected by default.
Org Type	TurboConfig, TurboPricing	Org type to be provisioned (sandbox or production)
Tenant Name	TurboPricing, TurboConfig	The one-word tenant name used for the tenant endpoint (for example, *customername-sandbox*)
Consumer Key	TurboConfig, TurboPricing	The consumer key (`client-id` in OAuth 2.0) generated from your Connected App. Refer to Creating a Connected App.
Consumer Secret	TurboConfig, TurboPricing	The secret key (`client-secret` in OAuth 2.0) generated from your Connected App. Refer to Creating a Connected App

Configuration	Required for Service	Description
Salesforce User Name	TurboConfig, TurboPricing	Admin username for the org to be provisioned with read/write access to Conga CPQ (used by Conga Technical Support for verifying settings)
Salesforce Password	TurboConfig, TurboPricing	Password for the Salesforce admin user.
Authority	TurboPricing	The URL used to verify session Id for TurboPricing (*login.salesforce.com*, *test.salesforce.com*, or a custom Salesforce domain)
InstanceURL	TurboPricing	The URL is given by the UI after logging into the org to be provisioned (for example, *customerturbo.my.salesforce.com*)
OAuthTokenURL	TurboConfig	Salesforce token endpoint URL (this will be *login.salesforce.com* or *test.salesforce.com*, depending on your Org Type)
Contact Email	TurboConfig, TurboPricing	The email to send the notifications on the successful provision.
Data Volume Settings	TurboConfig, TurboPricing	Provide the following entity information to Cloud Ops to perform the sizing accordingly. • Estimated Total Number of Search Filters • Estimated Total Number of Asset Line Items • Estimated Total Number of Deal Guidance Rules • Estimated Total Number of Price Rules • Estimated Total Number of Price List Items • Estimated Total Number of Price Matrixes • Estimated Total Number of Products (count of product records in product2 object)

Configuration	Required for Service	Description
Activate Multiple Currencies	TurboPricing	Verify if the *Activate Multiple Currencies* option is selected in the org. • If the option is selected, add **`"MultipleCurrenciesEnabled": true`** in the tenant configuration file with the help of DevOps/CloudOps, • If the option is not selected, add **`"MultipleCurrenciesEnabled": false`** in the tenant configuration file with the help of DevOps/CloudOps.

After collecting all the required information, provide it with your tenant provisioning request to Conga Technical Support.

Data Volume settings for Tenant Onboarding

Cloud Ops or Dev Ops must update the *OverrideSettings* object of onboarding API to handle the high volume data in the tenant org.

Handling the high-volume data in the tenant org is a difficult task when onboarding a new tenant. Therefore Cloud Ops or Dev Ops must update the *OverrideSettings* object of onboarding API.

This process is applicable only when onboarding the new tenant.

To update the O*verrideSettings* object

1. Collect the records count for the following entities and update the respective counts.

```
"OverrideSettings": {
    "APTTUS_APPROVAL__SEARCHFILTER__C": 10000,
    "APTTUS_CONFIG2__ASSETLINEITEM__C": 10000,
    "APTTUS_DEALMGR__DEALGUIDANCERULEENTRY__C": 10000,
    "APTTUS_CONFIG2__PRICERULEENTRY__C": 10000,
    "APTTUS_CONFIG2__PRICELISTITEM__C": 21000000,
    "APTTUS_CONFIG2__PRICEMATRIXENTRY__C": 10000,
    "PRODUCT2": 6100000
  }
```

2. Use the following API adding the updated *OverrideSettings* object in the body of the `“/pricing/dataadmin/onboardtenant”` API.

Post-Provisioning Tasks (TurboConfig)

The post-provisioning process for TurboConfig are divided into two main tasks:

- Configuring the service (custom settings and properties)
- Setting up and activating data sync for config master data from SFDC to TurboConfig

Refer to the following topics for step-by-step instructions to complete setup and configuration of TurboConfig:

- Configuring Remote Site Settings for TurboConfig
- Configuring TurboConfig Settings
- Configuring Custom Flows for TurboConfig

Configuring Remote Site Settings for TurboConfig

To create a Remote Site record

1. Go to **Setup** > **Administration Setup** > **Security Controls** > **Remote Site Settings**.
2. Click **New Remote Site**.
3. Enter the name in **Remote Site Name**. For example, *TurboConfig.*
4. Enter the URL for the remote site in the **Remote Site URL**.

> - Contact Conga Technical Support for the Remote Site URL.
> - Do not enter the '/' symbol at the end of the Remote Site URL.

5. Enable **Active**, if not selected by default.
6. Click **Save**.

Configuring TurboConfig Settings

You can enable TurboConfig either globally or for select CPQ flows. By default, the global **Contraint Rule Execution Mode** is set to *Client*. To enable the TurboConfig for select flows, refer to Configuring Custom Flows for TurboConfig. Follow the steps below to enable TurboConfig at the global level.

Take note of the following before you enable TurboConfig:

- Refer to Supported Features in TurboConfig topic to confirm that the features you use are supported in TurboConfig before enabling it globally.
- If you want to enable TurboConfig for certain types of quotes only, create a new flow with execution mode as *CMS*. Refer to Configuring Custom Flows for TurboConfig
- Once you create a quote using *CMS* execution mode you cannot switch to *Client* mode for that particular quote.
- The constraint rule execution mode (*Client* or *CMS*) is determined at the beginning of the quote. Once determined you cannot change them.
- The constraint rule execution mode *CMS* works on standalone products but the products must be published individually or in groups.

To enable TurboConfig on the quote

1. Go to **Setup** > **App Setup** > **Develop** > **Manage Custom Settings.**
2. Click **Config System Properties**. Click **Manage**.
3. Click **Edit** next to **System Properties**.
4. Define the fields as explained in the table below:

Field	**Description**
Constraint Rule Execution Mode	Enter the value *CMS* in the field.
CMS End Point URL	Enter the End Point URL for TurboConfig.

- CMS End Point URL is provided by Conga Technical Support.
- Do not enter the '/' symbol at the end of the CMS End Point URL.

5. Click **Save**.

Configuring Custom Flows for TurboConfig

You can enable TurboConfig for selective CPQ flows. You can use this functionality to avoid making TurboConfig as the default configuration engine and use the engine to process large and complex configuration rules. To enable TurboConfig for a specific flow, you must create a dedicated data set of **Config System Properties**.

To enable TurboConfig for specific flows

1. Create a custom flow. Refer to "Configuring Flow Settings" in *CPQ on Salesforce Administrator Guide*.

2. Create a formula action at the **Quote/Proposal** object for the flow you created in Step 1. Refer to "Creating Custom Buttons for Different Flows" in *CPQ on Salesforce Administrator Guide*.
3. Go to **Setup** > **App Setup** > **Develop** > **Custom Settings.**
4. Click **Config System Properties**. Click **Manage**.
5. Click **New** to create a new data set.
6. In the **Name** field, enter the name of the custom flow you created in Step 1.
7. Define **Constraint Rule Execution Mode** and **CMS End Point URL** as described in Step 4 in the topic Configuring TurboConfig Settings.
8. Click **Save**.

The Sales rep must use the custom flow that you created to configure the quote using TurboConfig.

Post-Provisioning Tasks (TurboPricing)

The post-provisioning process for TurboPricing are divided into four main tasks:

1. Configuring the service (custom settings and properties)
2. Configuring data sync service for use by administrators
3. Configure custom pricing callbacks
4. Setting up and activating data sync for pricing master data from SFDC to TurboPricing

Refer to the following topics for step-by-step instructions to complete setup and configuration of TurboPricing:

- Setting Up the TurboPricing Endpoint URL
- Setting Up the Pricing Execution Mode
- Configuring TurboEngines Callbacks

Setting Up the TurboPricing Endpoint URL

This section provides information for setting up the TurboPricing endpoint URL in the org.

1. Click the **All Tabs** icon (✚) and click **Admin**. The Home page is displayed.
2. Click **New**. The New Admin page is displayed.
3. In the **Name** field, enter *APTS_PricingServiceOverrideURl*.
4. In the **Value** field, enter the TurboPricing endpoint URL (without https://).
5. Click **Save**.

ⓘ Do not enter the forward-slash (/) symbol at the end of the Endpoint URL.

Setting Up the Pricing Execution Mode

This section provides information for setting up the Pricing Execution Mode in the org.

1. Go to **Setup** > **App Setup** > **Develop** > **Events** > **Custom Settings**.
2. Click **Config System Properties**. Click **Manage**.
3. Click **Edit** next to System Properties.
4. In the **Pricing Execution Mode**, enter *Turbo*.
5. Click **Save**.

Configuring TurboEngines Callbacks

This topic provides information on configuring TurboEngines callbacks.

Callbacks provide you with a mechanism to apply custom logic at the extension points provided and get invoked during the pricing computation by the TurboEngines. For example, you can apply custom pricing on the line items in the cart using the TurboEngines Callback Class. Callbacks are implemented using interfaces that are specific to each callback. These interfaces have various methods that you can use to achieve your task. You must implement the interface in a C# class and within that class, you may use your custom logic using the methods of the interface.

The sections in this topic provide information for:

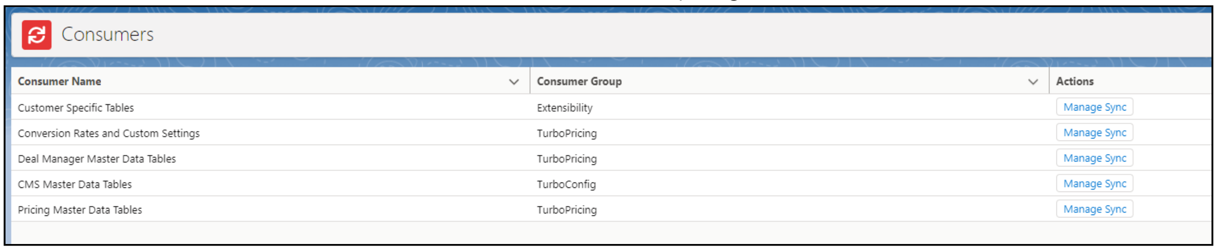
- Navigating the TurboEngines Callbacks Administrator User Interface
- Managing TurboEngines Callbacks
- Helper Functions for TurboEngines Callbacks
- Pricing Callback Class for TurboPricing

Navigating the TurboEngines Callbacks Administrator User Interface

This section provides information on navigating the TurboEngines callback administrator user interface.

1. Log in to the Salesforce org.
2. Click **Switch to Lightning Experience**.
3. Open the **Salesforce App Launcher (Lightning)** and launch the **TurboEngines Admin** app.

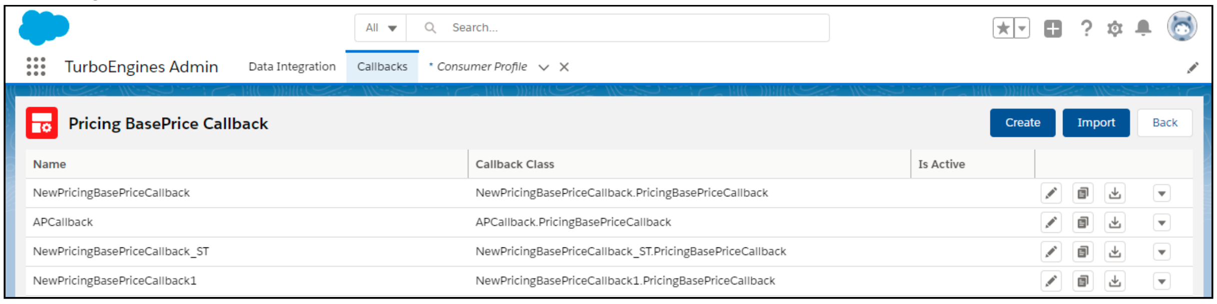
4. Click the **Callbacks** tab. A list of callbacks is displayed.



Consumer Name	Consumer Group	Actions
Customer Specific Tables	Extensibility	Manage Sync
Conversion Rates and Custom Settings	TurboPricing	Manage Sync
Deal Manager Master Data Tables	TurboPricing	Manage Sync
CMS Master Data Tables	TurboConfig	Manage Sync
Pricing Master Data Tables	TurboPricing	Manage Sync

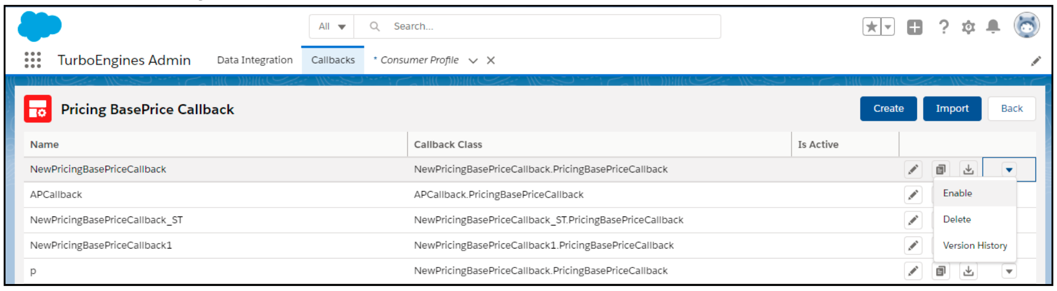
If a callback is enabled, the project name is displayed in the **Callback Name** column.

5. Click **Manage** under the **Actions** column for the required callback to configure it. The detail page of the selected callback with a list of projects related to the callback is displayed.



Name	Callback Class	Is Active
NewPricingBasePriceCallback	NewPricingBasePriceCallback.PricingBasePriceCallback	
APCallback	APCallback.PricingBasePriceCallback	
NewPricingBasePriceCallback_ST	NewPricingBasePriceCallback_ST.PricingBasePriceCallback	
NewPricingBasePriceCallback1	NewPricingBasePriceCallback1.PricingBasePriceCallback	

6. Click **Create** to create a new project.
7. Click **Import** to import an existing project.
8. Click one of the following icons for the required callback:
   - Edit ( )
   - Clone ( )
   - Download ( )
   - Click the Action icon ( ) and select one of the following options:
     - Enable or Disable
     - Delete
     - Version History



Name	Callback Class	Is Active
NewPricingBasePriceCallback	NewPricingBasePriceCallback.PricingBasePriceCallback	
APCallback	APCallback.PricingBasePriceCallback	
NewPricingBasePriceCallback_ST	NewPricingBasePriceCallback_ST.PricingBasePriceCallback	
NewPricingBasePriceCallback1	NewPricingBasePriceCallback1.PricingBasePriceCallback	
p	NewPricingBasePriceCallback.PricingBasePriceCallback	

9. From the Explorer panel, create a file under the current project if required. See Navigating the TurboEngines Callbacks Administrator User Interface.
10. Test your code before saving it. See Navigating the TurboEngines Callbacks Administrator User Interface.
11. Click the **Code Difference** icon ( ) to see the difference between the original and modified code. The differences are highlighted for easy identification.

12. At the bottom of the code editor panel:
    - Click the **Collapse** icon ( ) to show the panel and **Expand** icon ( ) to hide the panel.
    - **Output**: This tab displays the output of your code. Whenever you execute a method, the returned result is displayed in this tab.
    - **Input**: This tab displays the input of your code. You can verify what values CPQ has set for parameters or what values CPQ retrieves by reference for a parameter when you execute some code.
    - **Profiler**: This tab displays the order of execution of methods and performance of methods. You can also check how long CPQ takes to execute each method.
    - **Console**: This tab displays the log statements that you have added in the callback code when you test the callback code using the authoring UI.

Managing Files and Folders in the Explorer Panel of the Edit Project Page

This section describes how you can manage files and folders in the Explorer panel of the edit project screen. You can click the Collapse icon ( ) to show the panel and Expand icon ( ) to hide the panel.

Adding an Item

1. Select a folder if any and click the **Add Item** icon ( ). The Add Item pop-up is displayed.
2. From the Type drop-down, select what type of item you want to add. The supported values are File and Folder.
3. Enter a name for the item in the **Name** field.
4. Click **Create**.

Renaming an Item

1. Select a file to be renamed.
2. Click the **Rename Item** icon ( ). The Rename Item pop-up is displayed.
3. In the **Name** field, enter a new name and click **Save**.

Deleting an Item

1. Select a file to be removed.
2. Click **Delete Item** icon ( ). The Delete Item pop-up prompting you to confirm deletion is displayed.
3. Click **Yes** to delete the item.

If there is only one item, the Delete Item icon ( ) is disabled. You must have at least one item on the edit project page.

Testing a Callback Method by Executing the Code

CPQ allows you to test the code you have written for a callback method before saving it. In the **Test Run** panel, click the **Collapse** icon ( ) to show the panel and **Expand** icon ( ) to hide the panel.

1. From the **Class** drop-down, select a class. It displays all classes available in the project callback.
2. From the **Method** drop-down, select a method. It displays all methods currently available in the class. You can execute a method of that particular class. CPQ does not display private methods on the **Method** drop-down.
3. In the **Parameters** field, enter the code.
4. Click the **Refresh project metadata** icon ( ) to refresh project data. For example, if you have an unsaved method and want to execute it for validation, click
5. the **Refresh project metadata** icon. The new method is listed in the **Method** drop-down.
6. Click the **Execute** icon ( ) to execute the method.

Managing TurboEngines Callbacks

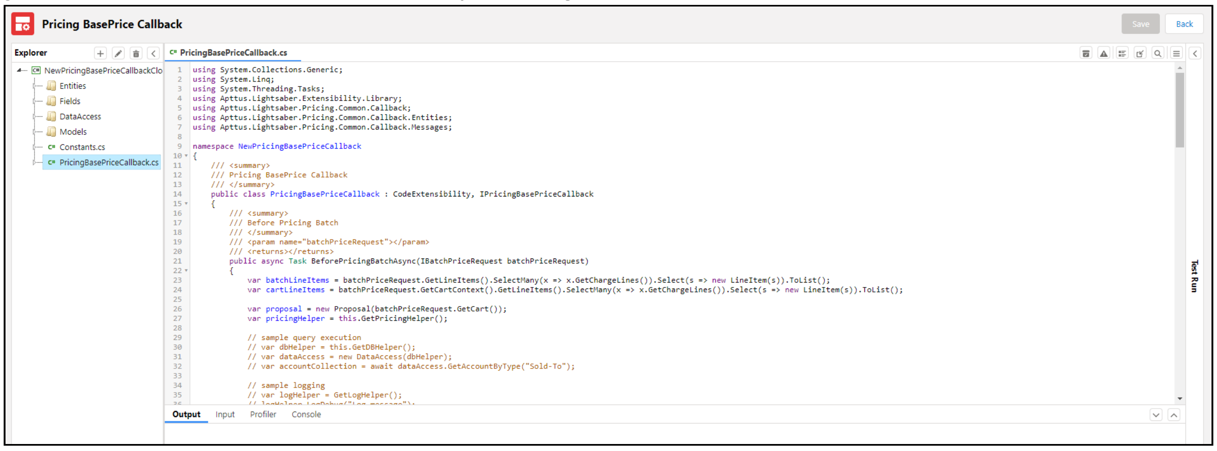
This section provides information on creating and importing callbacks and managing an existing TurboPricing callbacks.

- Creating a Callback
- Importing a Callback
- Editing a Callback
- Cloning a Callback
- Downloading a Callback
- Enabling a Callback
- Disabling a Callback
- Deleting a Callback
- Version History
- Validate QueryModel

Creating a Callback

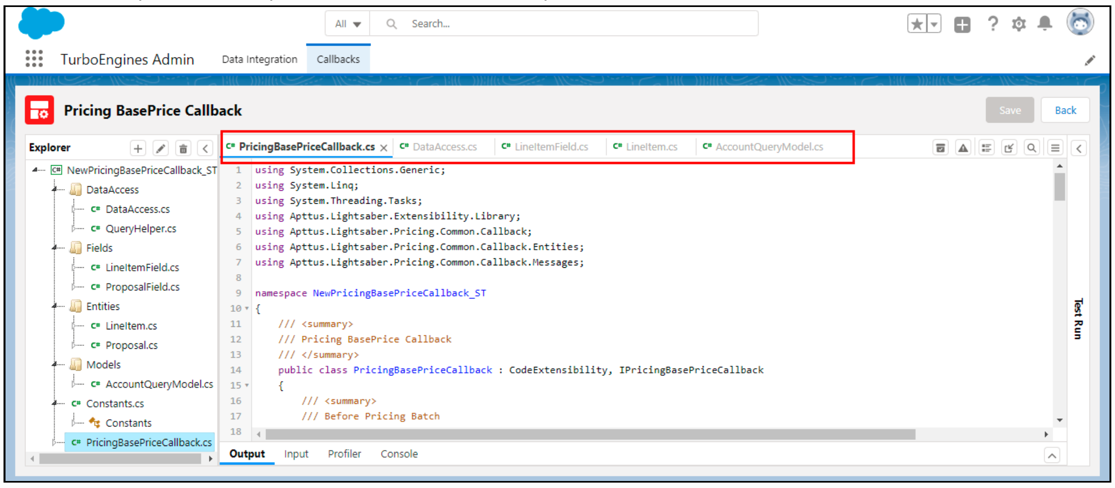
1. Log in to the Salesforce org.

2. Click **Switch to Lightning Experience**.
3. Open the **Salesforce App Launcher (Lightning)** and launch the **TurboEngines Admin** app.
4. Click the **Callbacks** tab. A list of callbacks is displayed. If a callback is enabled, the project name is displayed in the **Callback Name** column.
5. Click **Manage** under the **Actions** column for the required callback to configure it. The detail page of the selected callback with a list of projects related to the callback is displayed.
6. Click **Create** to create a new project. The New Callback pop-up is displayed.
7. Enter a name for the callback project and click Create. A new project is created and you are redirected to the edit project page.



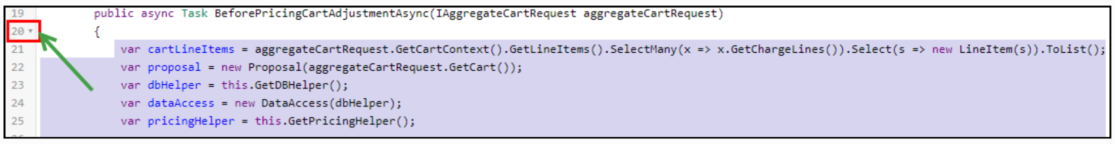
```
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Apttus.Lightsaber.Extensibility.Library;
using Apttus.Lightsaber.Pricing.Common.Callback;
using Apttus.Lightsaber.Pricing.Common.Callback.Entities;
using Apttus.Lightsaber.Pricing.Common.Callback.Messages;

namespace NewPricingBasePriceCallback
{
    /// <summary>
    /// Pricing BasePrice Callback
    /// </summary>
    public class PricingBasePriceCallback : CodeExtensibility, IPricingBasePriceCallback
    {
        /// <summary>
        /// Before Pricing Batch
        /// </summary>
        /// <param name="batchPriceRequest"></param>
        /// <returns></returns>
        public async Task BeforePricingBatchAsync(IBatchPriceRequest batchPriceRequest)
        {
            var batchLineItems = batchPriceRequest.GetLineItems().SelectMany(x => x.GetChargeLines()).Select(s => new LineItem(s)).ToList();
            var cartLineItems = batchPriceRequest.GetCartContext().GetLineItems().SelectMany(x => x.GetChargeLines()).Select(s => new LineItem(s)).ToList();

            var proposal = new Proposal(batchPriceRequest.GetCart());
            var pricingHelper = this.GetPricingHelper();

            // sample query execution
            // var dbHelper = this.GetDBHelper();
            // var dataAccess = new DataAccess(dbHelper);
            // var accountCollection = await dataAccess.GetAccountByType("Sold-To");

            // sample logging
            // var logHelper = GetLogHelper();
```

ⓘ You can open multiple files from the explorer in the code editor.



```
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Apttus.Lightsaber.Extensibility.Library;
using Apttus.Lightsaber.Pricing.Common.Callback;
using Apttus.Lightsaber.Pricing.Common.Callback.Entities;
using Apttus.Lightsaber.Pricing.Common.Callback.Messages;

namespace NewPricingBasePriceCallback_ST
{
    /// <summary>
    /// Pricing BasePrice Callback
    /// </summary>
    public class PricingBasePriceCallback : CodeExtensibility, IPricingBasePriceCallback
    {
        /// <summary>
        /// Before Pricing Batch
```

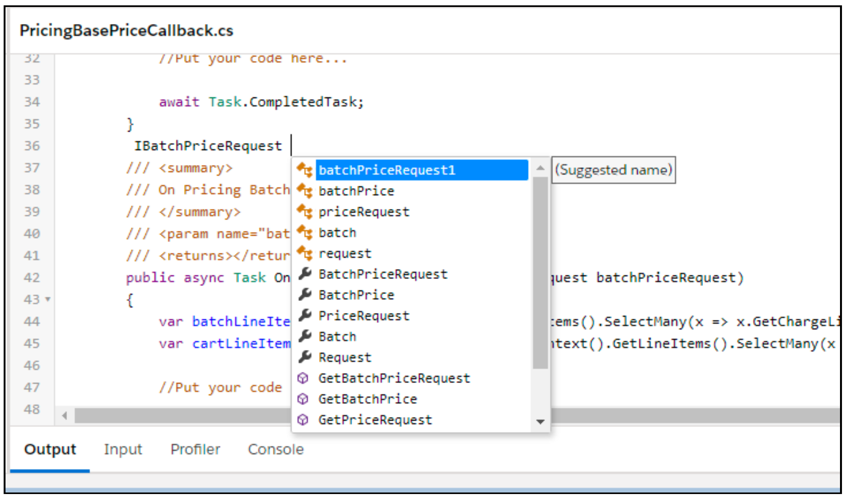
8. ⓘ You can collapse the code blocks by clicking the **Collapse** icon (▾).



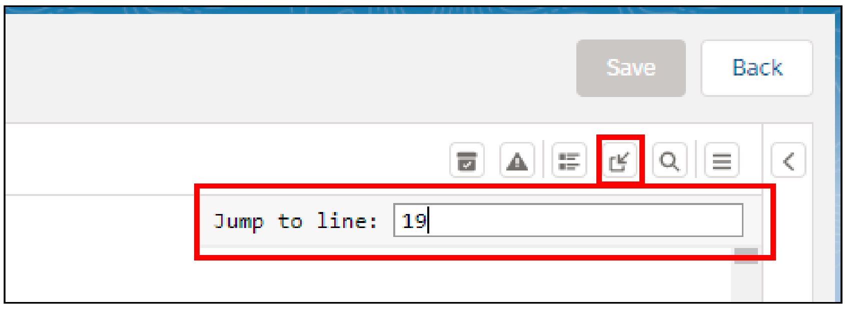
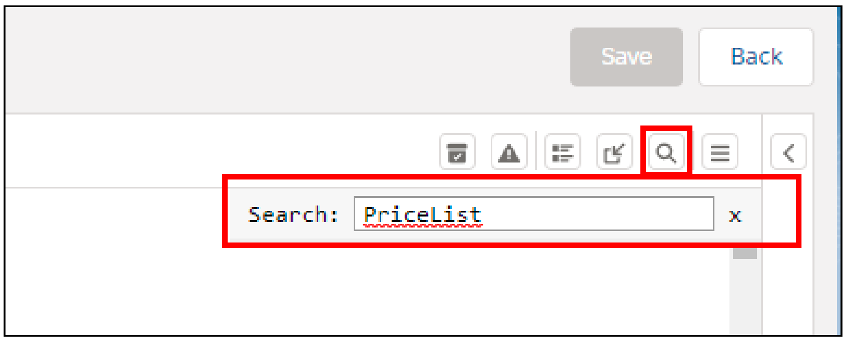
```
        public async Task BeforePricingCartAdjustmentAsync(IAggregateCartRequest aggregateCartRequest)
        {
            var cartLineItems = aggregateCartRequest.GetCartContext().GetLineItems().SelectMany(x => x.GetChargeLines()).Select(s => new LineItem(s)).ToList();
            var proposal = new Proposal(aggregateCartRequest.GetCart());
            var dbHelper = this.GetDBHelper();
            var dataAccess = new DataAccess(dbHelper);
            var pricingHelper = this.GetPricingHelper();
```

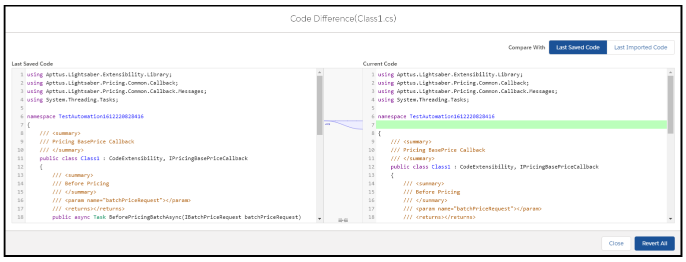
The following table describes the icons displayed on the code editor.

Icon or Feature	Name	Description

Auto Complete	Auto Complete	Press **Ctrl+Space** to view the auto-complete suggestions on the project code (C#) editor.  PricingBasePriceCallback.cs
		`//Put your code here...`
		`await Task.CompletedTask;`
		`}`
		`IBatchPriceRequest`
		`/// <summary>`
		`/// On Pricing Batch`
		`/// </summary>`
		`/// <param name="bat`
		`/// <returns></retur`
		`public async Task On` `uest batchPriceRequest)`
		`{`
		`var batchLineIte` `tems().SelectMany(x => x.GetChargeLi`
		`var cartLineItem` `ntext().GetLineItems().SelectMany(x`
		`//Put your code`
		batchPriceRequest1 (Suggested name)
		batchPrice
		priceRequest
		batch
		request
		BatchPriceRequest
		BatchPrice
		PriceRequest
		Batch
		Request
		GetBatchPriceRequest
		GetBatchPrice
		GetPriceRequest
		Output Input Profiler Console

☑	Validate QueryModel	To verify if the master entities or fields used in callback code are available in the extensibility data source or not.
⚠	Code Warnings	Displays the code warnings as per the C# guidelines at the bottom.
☰	Format	Formats and aligns the code (C#) in the editor as per C# coding standards.

Icon or Feature	Name	Description
	Go To Line (Ctrl+G)	To jump to the specific line in the editor.  Jump to line: 19
	Find	To search a particular term in the editor.  Search: PriceList x

	Code Difference	To compare and find the differences between *Last Saved Code or Last Imported Code* and *Current Code.*
		
		a. Click the icon. A **Code Difference** pop-up window is displayed with details of *Last Saved Code* and *Current Code.*
		
		Code Difference(Class1.cs)
		Compare With: Last Saved Code / Last Imported Code
		Last Saved Code
		`using Apttus.Lightsaber.Extensibility.Library;`
		`using Apttus.Lightsaber.Pricing.Common.Callback;`
		`using Apttus.Lightsaber.Pricing.Common.Callback.Messages;`
		`using System.Threading.Tasks;`
		
		`namespace TestAutomation1612220828416`
		`{`
		`    /// <summary>`
		`    /// Pricing BasePrice Callback`
		`    /// </summary>`
		`    public class Class1 : CodeExtensibility, IPricingBasePriceCallback`
		`    {`
		`        /// <summary>`
		`        /// Before Pricing`
		`        /// </summary>`
		`        /// <param name="batchPriceRequest"></param>`
		`        /// <returns></returns>`
		`        public async Task BeforePricingBatchAsync(IBatchPriceRequest batchPriceRequest)`
		Current Code
		`using Apttus.Lightsaber.Extensibility.Library;`
		`using Apttus.Lightsaber.Pricing.Common.Callback;`
		`using Apttus.Lightsaber.Pricing.Common.Callback.Messages;`
		`using System.Threading.Tasks;`
		
		`namespace TestAutomation1612220828416`
		
		`{`
		`    /// <summary>`
		`    /// Pricing BasePrice Callback`
		`    /// </summary>`
		`    public class Class1 : CodeExtensibility, IPricingBasePriceCallback`
		`    {`
		`        /// <summary>`
		`        /// Before Pricing`
		`        /// </summary>`
		`        /// <param name="batchPriceRequest"></param>`
		`        /// <returns></returns>`
		
		The code differences will be highlighted and can be identified easily.
		b. To compare the current code with the last imported code, click **Last Imported Code**.
		c. To retrieve the changes, click **Revert All**.

There is a sample template for you to create *IPricingBasePriceCallback*. Below is a basic implementation of *IPricingBasePriceCallback.* Once a project is created, it uses the callback template which has some pre-filled information for your reference. It enables the user to proceed with incorporating a custom logic.

```
using Apttus.Lightsaber.Extensibility.Library;
using Apttus.Lightsaber.Pricing.Common.Callback;
using Apttus.Lightsaber.Pricing.Common.Callback.Messages;
using System.Threading.Tasks;

namespace NewPricingBasePriceCallback_ST
{
    /// <summary>
    /// Pricing BasePrice Callback
    /// </summary>
     public class PricingBasePriceCallback : CodeExtensibility,
IPricingBasePriceCallback
     {
        /// <summary>
        /// Before Pricing
        /// </summary>
        /// <param name="batchPriceRequest"></param>
        /// <returns></returns>
        public async Task BeforePricingBatchAsync(IBatchPriceRequest
batchPriceRequest)
        {
            await Task.CompletedTask;
        }

        /// <summary>
        /// On Pricing
        /// </summary>
        /// <param name="batchPriceRequest"></param>
        /// <returns></returns>
        public async Task OnPricingBatchAsync(IBatchPriceRequest
batchPriceRequest)
        {
            await Task.CompletedTask;
        }

        /// <summary>
        /// After Pricing
        /// </summary>
        /// <param name="batchPriceRequest"></param>
        /// <returns></returns>
        public async Task AfterPricingBatchAsync(IBatchPriceRequest
batchPriceRequest)
        {
            await Task.CompletedTask;
        }
    }
}
```

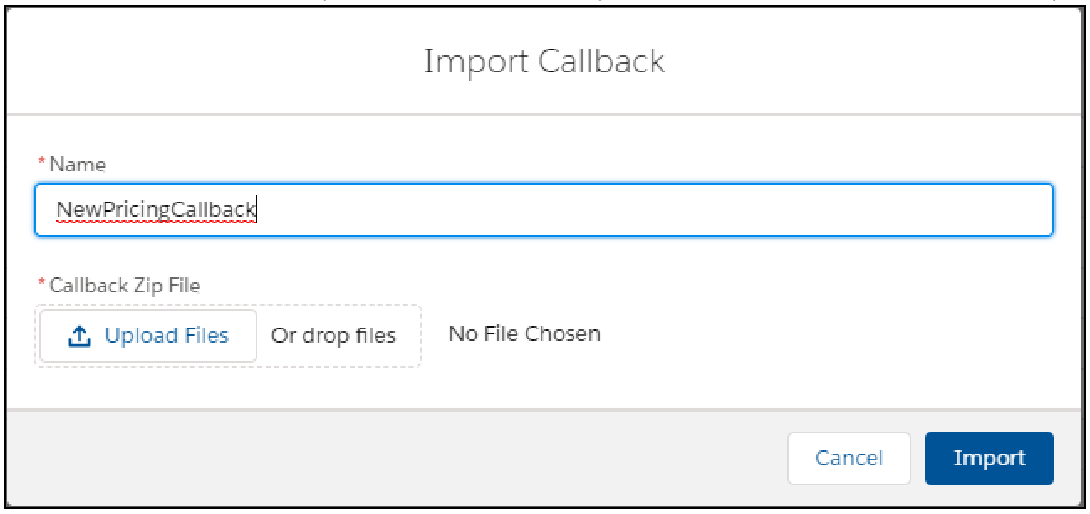
9. Create a callback using the template.
10. Click **Save**. If there are no errors, the code is saved successfully. If there is an error in the code, an error message is displayed. At the bottom of the editor panel, It displays the reason for the error. The user can click the error detail in the output tab and jump to a specific error line in the specific file. It helps the user in compiling the error.

Importing a Callback

The **Import** feature provides a mechanism to import the callback project (*.zip file*). Using this feature, you can move the callback code to a different environment if required.

1. Log in to the Salesforce org.
2. Click **Switch to Lightning Experience**.
3. Open the **Salesforce App Launcher (Lightning)** and launch the **TurboEngines Admin** app.
4. Click the **Callbacks** tab. A list of callbacks is displayed. If a callback is enabled, the project name is displayed in the **Callback Name** column.
5. Click **Manage** under the **Actions** column for the required callback to configure it. The detail page of the selected callback with a list of projects related to the callback is displayed.
6. Click **Import**. The Import Callback pop-up is displayed.
7. Enter a name for the callback project and click **Upload Files** or **drop files** to add the *.zip* file that you require to import.

8. Click **Import**. A new project is created and you are redirected to the edit project page.


Import Callback

* Name

NewPricingCallback

* Callback Zip File

Editing a Callback

1. Log in to the Salesforce org.
2. Click **Switch to Lightning Experience**.
3. Open the **Salesforce App Launcher (Lightning)** and launch the **TurboEngines Admin** app.
4. Click the **Callbacks** tab. A list of callbacks is displayed. If a callback is enabled, the project name is displayed in the **Callback Name** column.
5. Click **Manage** under the **Actions** column for the required callback to configure it. The detail page of the selected callback with a list of projects related to the callback is displayed.
6. Click the **Edit** icon (✎) for the required callback. The edit project page is displayed.
7. Perform the required edits to the project. See Creating a TurboPricing Callback.
8. Click **Save**.

Cloning a Callback

1. Log in to the Salesforce org.
2. Click **Switch to Lightning Experience**.
3. Open the **Salesforce App Launcher (Lightning)** and launch the **TurboEngines Admin** app.
4. Click the **Callbacks** tab. A list of callbacks is displayed. If a callback is enabled, the project name is displayed in the **Callback Name** column.
5. Click **Manage** under the **Actions** column for the required callback to configure it. The detail page of the selected callback with a list of projects related to the callback is displayed.

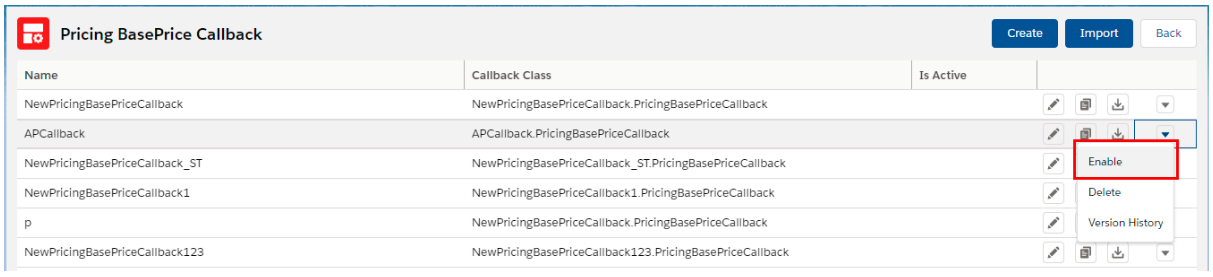
6. Click the **Clone** icon ( ) for the required callback. The New Callback pop-up is displayed. The name of the callback is <original_callback_name>Clone.
7. Enter a new name if you want.
8. Click **Clone**.

Downloading a Callback

1. Log in to the Salesforce org.
2. Click **Switch to Lightning Experience**.
3. Open the **Salesforce App Launcher (Lightning)** and launch the **TurboEngines Admin** app.
4. Click the **Callbacks** tab. A list of callbacks is displayed. If a callback is enabled, the project name is displayed in the **Callback Name** column.
5. Click the **Download** icon ( ) for the required callback. A <Callback Name>.zip file is downloaded to your system.

Enabling a Callback

1. Log in to the Salesforce org.
2. Click **Switch to Lightning Experience**.
3. Open the **Salesforce App Launcher (Lightning)** and launch the **TurboEngines Admin** app.
4. Click the **Callbacks** tab. A list of callbacks is displayed. If a callback is enabled, the project name is displayed in the **Callback Name** column.
5. Click **Manage** under the **Actions** column for the required callback to configure it. The detail page of the selected callback with a list of projects related to the callback is displayed.
6. Click the **Action** icon ( ) for the required callback and select **Enable**. The project is enabled.



Name	Callback Class	Is Active
NewPricingBasePriceCallback	NewPricingBasePriceCallback.PricingBasePriceCallback	
APCallback	APCallback.PricingBasePriceCallback	
NewPricingBasePriceCallback_ST	NewPricingBasePriceCallback_ST.PricingBasePriceCallback	
NewPricingBasePriceCallback1	NewPricingBasePriceCallback1.PricingBasePriceCallback	
p	NewPricingBasePriceCallback.PricingBasePriceCallback	
NewPricingBasePriceCallback123	NewPricingBasePriceCallback123.PricingBasePriceCallback	

ⓘ When you enable a project, CPQ directly impacts pricing engine runtime because of real-time updates.

Disabling a Callback

1. Log in to the Salesforce org.
2. Click **Switch to Lightning Experience**.
3. Open the **Salesforce App Launcher (Lightning)** and launch the **TurboEngines Admin** app.
4. Click the **Callbacks** tab. A list of callbacks is displayed. If a callback is enabled, the project name is displayed in the **Callback Name** column.
5. Click **Manage** under the **Actions** column for the required callback to configure it. The detail page of the selected callback with a list of projects related to the callback is displayed.
6. Click the Action icon (▼ for the required callback and select **Disable**. The project is disabled.

Deleting a Callback

1. Log in to the Salesforce org.
2. Click **Switch to Lightning Experience**.
3. Open the **Salesforce App Launcher (Lightning)** and launch the **TurboEngines Admin** app.
4. Click the **Callbacks** tab. A list of callbacks is displayed. If a callback is enabled, the project name is displayed in the **Callback Name** column.
5. Click **Manage** under the **Actions** column for the required callback to configure it. The detail page of the selected callback with a list of projects related to the callback is displayed.
6. Click the **Action** icon (▼) for the required callback and select **Delete**. The Delete Callback pop-up prompting you to confirm deletion is displayed. Click **Yes** to delete the callback.

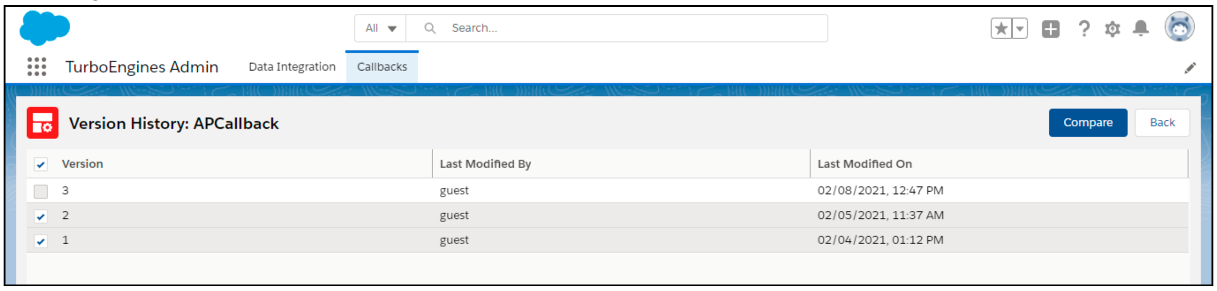
   ⓘ You cannot delete an enabled project. First, you need to disable the project and then you can delete it.

Version History

This feature allows the user to compare the different versions in order to identify the differences.

1. Log in to the Salesforce org.
2. Click **Switch to Lightning Experience**.
3. Open the **Salesforce App Launcher (Lightning)** and launch the **TurboEngines Admin** app.
4. Click the **Callbacks** tab. A list of callbacks is displayed. If a callback is enabled, the project name is displayed in the **Callback Name** column.

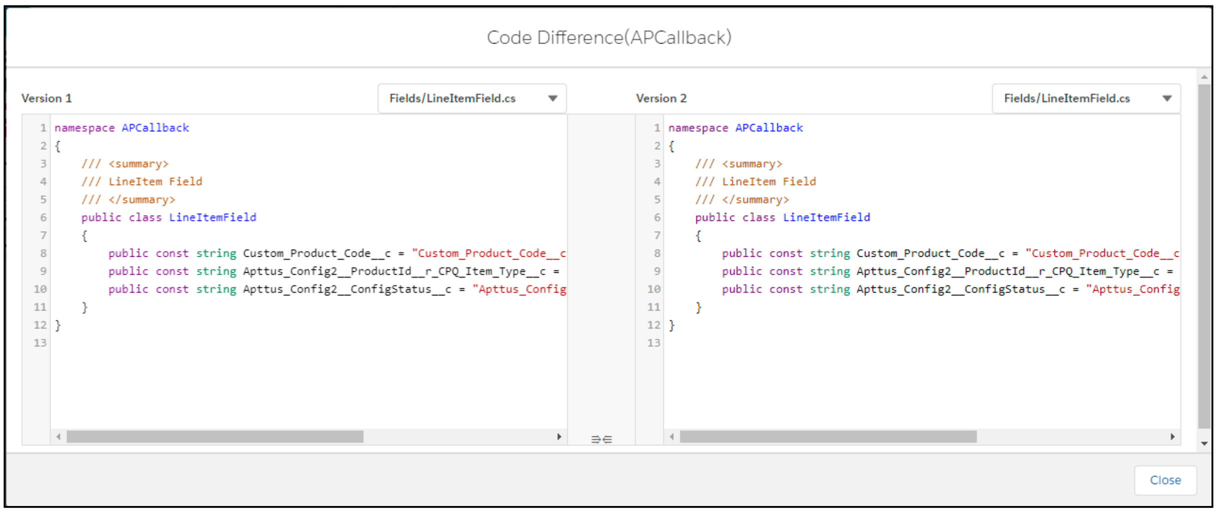
5. Click **Manage** under the **Actions** column for the required callback to configure it. The detail page of the selected callback with a list of projects related to the callback is displayed.
6. Click the **Action** icon (▼) for the required callback and select **Version History.** The detail page of the selected project with a list of versions related to the project is displayed.



Version History: APCallback

✓	Version	Last Modified By	Last Modified On
	3	guest	02/08/2021, 12:47 PM
✓	2	guest	02/05/2021, 11:37 AM
✓	1	guest	02/04/2021, 01:12 PM

7. Click the checkbox of the desired version and click **Compare**. The code difference of the selected callback project is displayed.



Code Difference(APCallback)

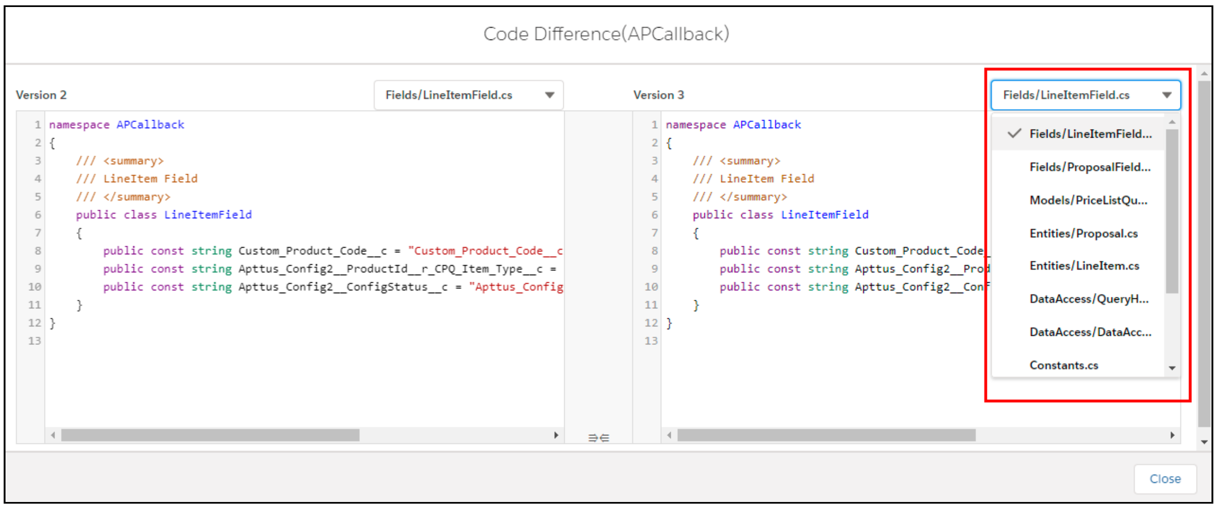
Version 1 — Fields/LineItemField.cs

```
namespace APCallback
{
    /// <summary>
    /// LineItem Field
    /// </summary>
    public class LineItemField
    {
        public const string Custom_Product_Code__c = "Custom_Product_Code__c
        public const string Apttus_Config2__ProductId__r_CPQ_Item_Type__c =
        public const string Apttus_Config2__ConfigStatus__c = "Apttus_Config
    }
}
```

Version 2 — Fields/LineItemField.cs

```
namespace APCallback
{
    /// <summary>
    /// LineItem Field
    /// </summary>
    public class LineItemField
    {
        public const string Custom_Product_Code__c = "Custom_Product_Code__c
        public const string Apttus_Config2__ProductId__r_CPQ_Item_Type__c =
        public const string Apttus_Config2__ConfigStatus__c = "Apttus_Config
    }
}
```

8. In the Code Difference popup window, click the dropdown available at the top of the code editor to change the code (*.cs) type.



Code Difference(APCallback)

Version 2

```
namespace APCallback
{
    /// <summary>
    /// LineItem Field
    /// </summary>
    public class LineItemField
    {
        public const string Custom_Product_Code__c = "Custom_Product_Code__c
        public const string Apttus_Config2__ProductId__r_CPQ_Item_Type__c =
        public const string Apttus_Config2__ConfigStatus__c = "Apttus_Config
    }
}
```

Version 3

```
namespace APCallback
{
    /// <summary>
    /// LineItem Field
    /// </summary>
    public class LineItemField
    {
        public const string Custom_Product_Code_
        public const string Apttus_Config2__Prod
        public const string Apttus_Config2__Conf
    }
}
```

Validate QueryModel

This feature allows the user to verify if the master entities or fields used in callback code are available in the extensibility data source or not.

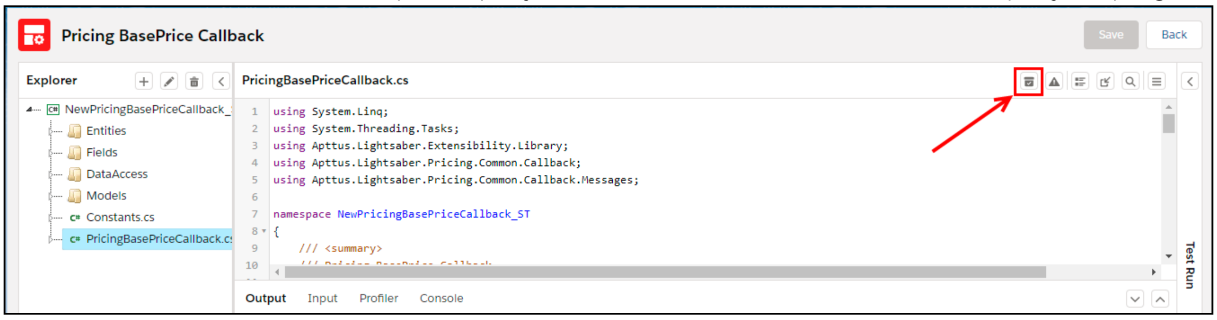
The user should follow such guidelines in order to use this feature accurately.

1. The QueryModel class should be decorated with the attribute [Entity="name of the entity"].
2. All the fields (select clause, where clause etc) used in the query must be specified in QueryModel class with the correct API names".

To verify the master entities or fields:

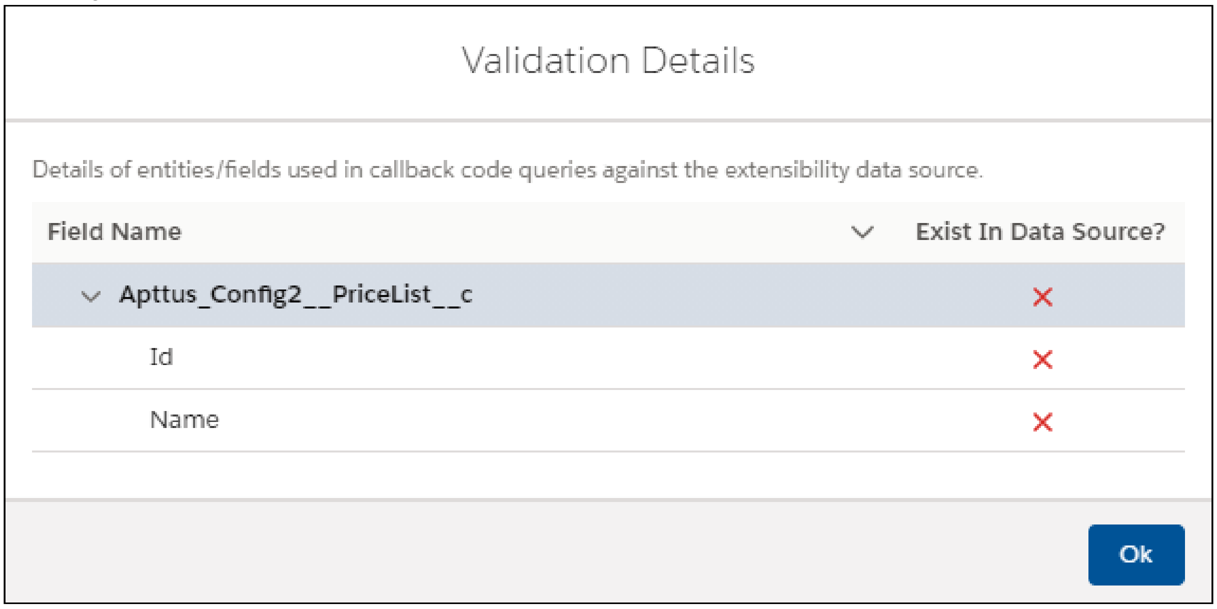
1. Log in to the Salesforce org.
2. Click **Switch to Lightning Experience**.
3. Open the **Salesforce App Launcher (Lightning)** and launch the **TurboEngines Admin** app.
4. Click the **Callbacks** tab. A list of callbacks is displayed. If a callback is enabled, the project name is displayed in the **Callback Name** column.
5. Click **Manage** under the **Actions** column for the required callback to configure it. The detail page of the selected callback with a list of projects related to the callback is displayed.

6. Click the Edit icon for the required project. You are redirected to the edit project page.


Pricing BasePrice Callback
Save
Back
Explorer
NewPricingBasePriceCallback_
Entities
Fields
DataAccess
Models
Constants.cs
PricingBasePriceCallback.cs
PricingBasePriceCallback.cs
Output
Input
Profiler
Console
Test Run

```
using System.Linq;
using System.Threading.Tasks;
using Apttus.Lightsaber.Extensibility.Library;
using Apttus.Lightsaber.Pricing.Common.Callback;
using Apttus.Lightsaber.Pricing.Common.Callback.Messages;

namespace NewPricingBasePriceCallback_ST
{
    /// <summary>
```

7. Click the **Validate QueryModel** ( ) icon. The Validation Details popup window is displayed.



Validation Details

Details of entities/fields used in callback code queries against the extensibility data source.

Field Name	Exist In Data Source?
Apttus_Config2__PriceList__c	✗
Id	✗
Name	✗

Helper Functions for TurboEngines Callbacks

This section describes the Helper functions available for TurboPricing callbacks.

- DBHelper
- LogHelper
- PricingHelper
- MetadataHelper
- VaultHelper
- HttpHelper
- CacheHelper

DBHelper

The DBHelper function helps you fetch the required documents from the database based on Query and *FilterConditions*, and fields that you require from the documents that fulfill the conditions. You can get the instance of *IDBHelper* by calling the method *GetDBHelper()* and it provides you with the following methods:

Methods

```
Task<List<T>> FindAsync<T>(Query query);
Task<List<T>> FindAsync<T>(string entityName, Expression<Func<T, bool>>
filterCondition, int? limit, params string[] fields);
Task<List<T>> FindAsync<T>(string entityName, Expression<Func<T, bool>>
filterCondition, params string[] fields);
DBStatisticsInfo GetDBStatistics();
```

The first method helps you fetch the documents based on a *Query*, which comprises of the *EntityName(Name of the collection)*, list of *FilterConditions*, *Fields* to fetch, and *Limit* of results to retrieve.

The *GetDBStatistics()* provides you the information on the total number of queries and the time taken for each query during callback execution.

Note

GetDBStatistics() method should be used only when you are debugging or troubleshooting the issue. Once the debugging/troubleshooting is done, you must remove it from your callback code.

Definitions

```
public class Query
{
    ExpressionOperator ExpressionOperator = ExpressionOperator.AND;
    public string EntityName { get; set; }
    public List<FilterCondition> Conditions { get; set; }
    public Expression Criteria {get; set;}
    public string[] Fields { get; set; }
    public int? Limit { get; set; }
}

public class FilterCondition
{
    public string FieldName { get; set; }
    public ConditionOperator ComparisonOperator { get; set; }
    public object Value { get; set; }
}

public class Expression
{
    public ExpressionOperator ExpressionOperator { get; set; }
    public List<FilterCondition> Conditions { get; }
    public List<Expression> Filters { get; }
}

ComparisonOperator :
    EqualTo,
    GreaterThan,
    GreaterThanOrEqualTo,
    In,
    LessThan,
    LessThanOrEqualTo,
    NotEqualTo,
    NotIn,
    Contains
```

- *EntityName* is the name of the collection you want to fetch the records from, or the records of the entity that you want to query upon.
- *Conditions* are a list of *FilterConditions*, which compare a field or a column's value with a specified *Value*.
- *Fields* are the columns of the entity you want in your query results.
- *Limit* is the number of records you want to limit your result to.

The following are a few examples of different Query Models: The following are a few examples of executing queries using DBHelper. The examples are for your reference only and may not execute as is.

Simple Query Example

```
Query query = new Query()
{
  EntityName = "Account",
  Fields = new string[]{ "Id", "Name", "Type" },
  Limit = 5,
  Conditions = new List<FilterCondition>()
  {
    new FilterCondition()
    {
      ComparisonOperator = ConditionOperator.EqualTo,
      FieldName = "Type", Value = "Ship-To"
    }
  }
};
public class AccountQueryModel
{
public string Id {get; set;}
public string Name {get; set;}
public string Type {get; set;}
}
```

DB Call Example

```
var dbHelper = GetDBHelper();
List<AccountQueryModel> account = await dbHelper.FindAsync<AccountQueryModel>(query);
```

Output

After you execute *FindAsync,* it retrieves data from the database and returns a list of AccountQueryModel.

Complex Query Example

```
Query query = new Query()
{
   EntityName = "Account",
   Fields = new string[]{ "Id", "Name", "Type","BillingCity__c",
"BillingCountryCode__c" },
   Limit = 2,
   Criteria = new Expression(ExpressionOperator.AND)
};

FilterCondition nestedConditionOne = new FilterCondition()
{
     ComparisonOperator = ConditionOperator.EqualTo,
     FieldName = "Type",
     Value = "Sold-To"
};
query.Criteria.AddCondition(nestedConditionOne);

Expression complexExpression = new Expression(ExpressionOperator.OR);
FilterCondition complexExpressionConditionOne = new FilterCondition()
{
     ComparisonOperator = ConditionOperator.In,
     FieldName = "BillingCity__c",
     Value =  new string[] { "BEDFORD PARK", "FREMONT" }
};
FilterCondition complexExpressionConditionTwo = new FilterCondition()
{
     ComparisonOperator = ConditionOperator.EqualTo,
     FieldName = "BillingCountryCode__c",
     Value =  "US"
};
complexExpression.AddCondition(complexExpressionConditionOne);
complexExpression.AddCondition(complexExpressionConditionTwo);
query.Criteria.AddFilter(complexExpression);
```

DB Call Example

```
var dbHelper = GetDBHelper();
List<AccountQueryModel> accountList = await
dbHelper.FindAsync<AccountQueryModel>(query);
return account;
```

Result

After you execute *FindAsync,* it retrieves data from the database and returns a list of AccountQueryModel.

Note:

For better performance, consider the following scenarios when executing database queries.

1. Apply more filters to narrow down the search results.
2. Fetch only the required columns or fields.
3. Do not execute queries inside *for* loop.
4. Create appropriate indexes based on the fields used in the query. You can create indexes on an entity using datasync UI.

LogHelper

The LogHelper function helps you with the tracing or logging. The *ILogHelper* instance is returned when *GetLogHelper()* is called provides you with the following methods.

Methods Available in LogHelper

```
void LogCritical(string message, Exception ex);
void LogDebug(string message);
void LogDebug(object obj);
void LogError(string message, Exception ex);
void LogInformation(string message);
void LogTrace(string message);
void LogWarning(string message);
```

Example

```
var logHelper = GetLogHelper();
logHelper.LogTrace("Trace Statement");
logHelper.LogDebug("Debug Statement");
logHelper.LogWarning("Warning Statement");
logHelper.LogInformation("Info Statement");
AccountQueryModel account = new AccountQueryModel(){
  Id= "0013i0000043m9eAAA",
  Name =  "Account_B5F83C"
};
logHelper.LogDebug(account);
logHelper.LogError("Error Message", new Exception("SystemException"));

public class AccountQueryModel
{
  public string Id {get; set;}
  public string Name {get; set;}
  public string Type {get; set;}
}
```

Console Output

```
Trace Statement
Debug Statement
Warning Statement
Info Statement
{"Id":"0013i0000043m9eAAA","Name":"Account_B5F83C","Type":null}
Error Message
{
  "ClassName": "System.Exception",
  "Message": "SystemException",
  "Data": null,
  "InnerException": null,
  "HelpURL": null,
  "StackTraceString": null,
  "RemoteStackTraceString": null,
  "RemoteStackIndex": 0,
  "ExceptionMethod": null,
  "HResult": -2146233088,
  "Source": null,
  "WatsonBuckets": null
}
```

When debugging or troubleshooting callback related issues, follow the minimal logging approach and add log statements in the callback code and then remove log statements from the code once it is done.

PricingHelper

The PricingHelper function helps you perform pricing-related operations such as apply rounding and updating the Price Method for line items. To get the instance of *IPricingHelper*, you must call *GetPricingHelper()*, which provides the following methods:

Methods

```
decimal? ApplyRounding(decimal? value, int precision, RoundingMode roundingMode);
void UpdatePrice(ILineItemModel lineItemModel);
public decimal? ApplyRounding(decimal? value, int precision, RoundingMode
roundingMode);
public decimal? ApplyRounding(decimal? value, int? currencyPrecision = null);
public void UpdatePrice(ILineItemModel lineItemModel);
public bool AddPricePointAdjustmentValueToPriceWaterfall(ILineItemModel lineItem,
IPricePointAdjustmentValue newPricePointAdjValue, int? index = null);
public IPricePointAdjustmentValue CreatePricePointAdjustmentValue();
public void RecalculatePriceWaterfall(ILineItemModel lineItem);
public bool RemoveAddedAdjFromPriceWaterfall(ILineItemModel lineItem, bool
recalculate, string pricePoint, PricePointAdjustmentType type, string name, string
sourceId);
public IConfigSystemProperties GetSystemProperties();
public Task<IList<T>> GetRecordsAsync<T>(IQueryObject queryObject);
public IQueryObject CreateQueryObject();
public T CreateEntity<T>() where T : IBaseEntity;
```

Example

```
var pricingHelper = GetPricingHelper();
var logHelper = GetLogHelper();

var roundedValue = pricingHelper.ApplyRounding(1.657m,2,RoundingMode.DOWN);
logHelper.LogDebug(roundedValue);

roundedValue = pricingHelper.ApplyRounding(1.657m,2,RoundingMode.UP);
logHelper.LogDebug(roundedValue);
```

Console Output

```
1.65
1.66
```

MetadataHelper

The MetadataHelper function helps you fetch the metadata for an entity for a specified RecordType. The *IMetadataHelper* instance returned when *GetMetadataHelper()* is called performs the following method:

Method

```
Task<string> GetRecordTypeIdbyName(string entityName, string recordTypeName);
```

Example

```
var metadataHelper = GetMetadataHelper();
var logHelper = GetLogHelper();

var recordTypeId = await metadataHelper.GetRecordTypeIdbyName("Product2", "Master");
logHelper.LogDebug(recordTypeId);
```

Console Output

```
012000000000000AAA
```

VaultHelper

The VaultHelper function helps you fetch the KeyVault details for an application. The *IVaultHelper* instance returned when *GetVaultHelper()* is called fetches KeyVault values using the following method:

Method

```
Task<Dictionary<string, object>> GetValue(string AppName);
```

Example

```
var vaultHelper = GetVaultHelper();
var logHelper = GetLogHelper();

var vault = await vaultHelper.GetValue("MyApp");
logHelper.LogDebug(vault["AppKey"]);
logHelper.LogDebug(vault["AppSecrets"]);
```

Console Output

```
"Your_Key"
"Your_Secrets"
```

To set and update value in vault

Using the Vault API, you can set and update value in vault, which is stored in encrypted format. You cannot retrieve vault value using any API. You can retrieve vault values only using the VaultHelper function.

1. **Create:** POST: /pricing/api/admin/Vault
   It creates an application with the specified key value name and application name.
   Body:

   ```
   {
       "AppName": "MyApp",
       "Secrets": {
           "MySecretsKey1" : "MyValue1",
           "MySecretsKey2" : "MyValue2"
       }
   }
   ```
   You can create multiple applications and store multiple secrets as KeyValue in a single application using the POST API and retrieve those secrete values using the VaultHelper function by passing AppName.
2. **Update:** PUT /pricing/api/admin/Vault/{appName}
   It updates an existing application vault information with a new value.
   Body:

```
{
    "AppName": "MyApp",
    "Secrets": {
        "MySecretsKey1" : "MyNewValue1",
        "MySecretsKey2" : "MyNewValue2"
    }
}
```

3. **Remove:** DELETE /pricing/api/admin/Vault/{appName}

   It removes secretes from the vault storage.

HttpHelper

The HttpHelper function helps you with HTTP calls covering CRUD operations.

To get an instance of the *IHttpHelper*, you must call *GetHttpHelper()*, which returns you an instance of *IHttpHelper* allowing you to call the following methods:

> It is not recommended to call salesforce API from Custom Code, due to performance reasons.

Methods

```
HttpContent GetHttpContenFromtXml<T>(T payload);
Task<T> GetAsync<T>(string requestUri);
Task<HttpResponseMessage> GetAsync(string requestUri);
Task<T> PostAsync<T>(string requestUri, HttpContent content);
Task<HttpResponseMessage> PostAsync(string requestUri, HttpContent content);
Task<T> PutAsync<T>(string requestUri, HttpContent content);
Task<HttpResponseMessage> PutAsync(string requestUri, HttpContent content);
Task<HttpResponseMessage> DeleteAsync(string requestUri);
```

Example: Invoke GET API call

```
var httpHelper = GetHttpHelper();
var logHelper = GetLogHelper();

string url = "https://jsonplaceholder.typicode.com/todos/1";
var response = await httpHelper.GetAsync<SampleResponse>(url);
logHelper.LogDebug(response);
```

Console Output

```
Complete-GetAsync(https://jsonplaceholder.typicode.com/todos/1), HttpStatus=OK,
Time=285.534
{"userId":1,"id":1,"title":"delectus aut autem","completed":false}
```

Example: Invoke POST API call

```
var httpHelper = GetHttpHelper();
var logHelper = GetLogHelper();

string url = "https://jsonplaceholder.typicode.com/todos/1";
var response = await httpHelper.GetAsync<SampleResponse>(url);

url = "https://jsonplaceholder.typicode.com/todos";
response.id = 0;
response.title = "Callback";
var contentToPost = new
 StringContent(JsonConvert.SerializeObject(response),Encoding.UTF8,"application/json")
;
var postResult = await httpHelper.PostAsync<SampleResponse>(url,contentToPost);
logHelper.LogDebug(postResult);
```

Console Output

```
Complete-GetAsync(https://jsonplaceholder.typicode.com/todos/1), HttpStatus=OK,
Time=285.534
{"userId":1,"id":1,"title":"delectus aut autem","completed":false}
Complete-PostAsync(https://jsonplaceholder.typicode.com/todos), HttpStatus=Created,
Time=400.073
{"userId":1,"id":201,"title":"Callback","completed":false}
```

CacheHelper

The CacheHelper function helps you cache data for faster retrievals. The *ICacheHelper* instance, which is returned when *GetCacheHelper()* is called, provides you with the following methods:

Methods

```
void Set<T>(string key, T value, string regionName);
T Get<T>(string key, string regionName);
bool Contains(string key, string regionName);
```

Example

```
public class Product2
{
  public string Id {get; set;}
  public string Name { get; set;}
}
```

Set Data in Cache

```
var cacheHelper = GetCacheHelper();

Product2 product = new Product2()
{
  Id= "RT001", Name =  "Router"
};
await cacheHelper.Set<Product2>("Product",product);
```

Console Output

```
{"Id":"RT001","Name":"Router"}
```

Get Data from Cache

```
var cacheHelper = GetCacheHelper();
var logHelper = GetLogHelper();

var productFromCache = await cacheHelper.Get<Product2>("Product");
logHelper.LogDebug(productFromCache);
```

Console Output

```
{"Id":"RT001","Name":"Router"}
```

Pricing Callback Class for TurboPricing

Pricing callback provides extensibility points in the Pricing Engine which can be used to extend or override existing behavior of the Pricing Engine based on customer requirements. Pricing callback classes allow you to add pricing logic to the cart that cannot be achieved by out-of-the-box pricing mechanisms, such as Price Rulesets and Price Matrices.

To use the Pricing Callback you must create a custom C# class that implements the following interfaces

Interface	Description
IPricingBasePriceCallback	This interface provides you a mechanism to define custom logic to be executed before, during, and after Base Price Calculation
IPricingTotallingCallback	This interface provides you a mechanism to define custom logic to be executed before, during, and after adjustment Calculation
IRelatedPricingCallback	This interface provides you a mechanism to define custom logic for calculating the pricing for related product line items

For accessing non-transactional/master data in callback code, you must only use the data synced in the Extensibility consumer profile. You can use the *DBHelper* functions to retrieve the synced master data from the TurboPricing data source. For accessing any transactional data such as line-item custom fields, proposal fields, cart fields, and so on, the line item custom fields should be configured in the custom setting "Configure LineItem Custom Fields" and Proposal/cart fields should be configured in the custom setting "Cart Header Criteria Fields" at Salesforce. Also, the line item's product or option relationship fields used in callback must be configured in the Price List Item entity as a part of the "Pricing Master Data Tables" consumer profile.

The following sections describe the interfaces:

- Coding Guidelines and Best Practices for TurboEngines Callbacks
- Pricing Base Price Callback Interface
- Pricing Totalling Callback Interface
- Related Pricing Callback Interface
- Migrating Salesforce Pricing Callback to TurboEngines Pricing Callback

Coding Guidelines and Best Practices for TurboEngines Callbacks

This section provides guidelines to follow while coding the TurboEngines compatible callbacks.

General Coding Guidelines for TurboEngines Callbacks

Guideline	Sub-topic	Description
Naming Conventions		
	Naming a Class	• Use Pascal casing standard for naming a class. • For example, the class name should be *PricingBasePriceCallback* instead of *pricingBasePriceCallback.*
	Naming a Method	• Use Pascal casing standard for naming a method. • For example, the method name should be *BeforePricingCartAdjustmentAsync* instead of *beforePricingCartAdjustmentAsync.*
	Naming a variable or method parameter	• Use Camel casing standard for naming a variable or method parameter. • For example, the variable name should be cartLineItems instead of CartLineItems. • Use meaningful, descriptive names for naming a variable or method parameter. • For example, the variable name should be cartLineItems instead of crtLnItms
	Naming a class property	• Use Pascal casing standard for naming a class property • For example, the property name should be Id instead of id • Use meaningful, descriptive names for naming the property • For example, the property name should be IsPrimaryLine instead of IsPrmryLn.

Coding Guidelines for TurboEngines Callbacks

You must follow the below guidelines while writing the callback code (C#) for TurboEngines.

When you create a new callback project using TurboEngines callback authoring UI, the sample callback project is auto-generated by the TurboEngine Extensibility framework as per the TurboEngine Callback coding guidelines. You can enhance it further with a custom implementation.

Guideline	Description

Using Statements	1. When you are writing the callback code:
	a. Add all the required using statements for the callback code.
	b. Below are the supported using statements in TurboEngine C# callback code:
	i. .NET core framework provided using statements
	ii. Newtonsoft.json provided using statements
	iii. TurboEngine Extensibility Framework provided using statements.
	c. For your reference, TurboEngine Extensibility Framework provides using statements are as follows. You must use the required using statements.
	`using Apttus.Lightsaber.Pricing.Common.Callback;`
	`using Apttus.Lightsaber.Pricing.Common.Callback.Entities;`
	`using Apttus.Lightsaber.Pricing.Common.Callback.Messages;`
	`using Apttus.Lightsaber.Pricing.Common.Callback.Enums;`
	`using Apttus.Lightsaber.Pricing.Common.Callback.Models;`
	`using Apttus.Lightsaber.Extensibility.Library;`
	`using Apttus.Lightsaber.Extensibility.Library.Attributes;`
	`using Apttus.Lightsaber.Extensibility.Library.Extension;`
	`using Apttus.Lightsaber.Extensibility.Library.Interface;`
	`using Apttus.Lightsaber.Extensibility.Library.Model;`

Namespace naming	1. Namespace's name must have the following convention: **Apttus.Lightsaber.{OrganizationName}.{OtherDetails}** a. For example, when writing a pricing callback implementation for Totaling callback for organization X, write the namespace as following - **Apttus.Lightsaber.X.Totalling**

Guideline	Description
Class Naming	1. The main class implementing the callback interface must always inherit from **CodeExtensibility**. 2. The naming of the main class must be based on the interface name. Basically, remove "I" prefix from the interface name. a. For example, pricing callback implementation for Totaling callback must implement *IPricingTotallingCallback.* Hence, the class name must be *PricingTotallingCallback.*

Methods	1. Methods should follow async/await pattern implementation.
	    a. Following is an example method representation without async/ await implementation.
	`public void MethodName() {`
	`    return;`
	`}`
	    b. Following is an example method implementation with async/await implementation.
	`public async Task MethodName() {`
	`    await Task.CompletedTask;`
	`}`

Variables	• For better precision, use decimal datatype instead of double datatype.

Guideline	Description
Database queries related	1. All the database query invocation must be part of the class named **DataAccess** under the folder DataAccess. 2. Ensure that all the DB queries for fetching the master data records from TurboEngine Extensibility data source (For example, MongoDB) are written in **QueryHelper** under the DataAccess folder. a. This provides a reference and a single place to identify all the queries used by a callback implementation. 3. To hold the result/response of the query execution for each of the queries executed, the respective model class needs to be created as follows. A sample model class is already provided in the auto-generated callback project when you create a new callback project. a. You must create all the model classes under the **Models** folder. b. The name of the class must end with QueryModel. i. For example, the model class to hold the result/response of Account query execution should be named as AccountQueryModel. c. Ensure that the field names of QueryModel match the field names used in the query. d. Ensure that the QueryModel class is annotated with the Entity attribute. The entity name should be spelled the same as it is in Salesforce. i. For example, The AccountQueryModel class should have the Entity attribute added as **[Entity("Account")]**. Here Account is the entity name as available in Salesforce.
Constants	• Ensure that all the general-purpose constants used in the callback implementation are specified in the Constants class.
Custom Fields	• The classes under the Fields folder must hold the field name constants for any custom fields used in the callback implementation. These class names must end with Field. • For example, if the class to hold the proposal object's custom field must be ProposalField. • Generally, you are required to create such classes for the line item, proposal, and cart object fields.

Guideline	Description
LineItem and Proposal entity model objects	• To easily access LineItem and Proposal fields for getting or setting the values in the callback implementation, LineItem and Proposal class are auto-created under the Entities folder by the TurboEngine Extensibility framework ◦ You must use those classes in your callback implementation and add the custom field details to it as per your implementation. ◦ When you are adding custom field details to LineItem or Proposal class, ▪ The relationship field must be provided as shown in the following example. For example, If your relationship field is **Apttus_Config2__ProductId__r.APTS_field__c** then the field name is **Apttus_Config2__ProductId__r_APTS_field__c.** Here "." is replaced with "_" in the name. ▪ In getter methods for the fields, use *GetLookupValue<T>()* method for relationship fields and use *GetValue<T>()* method for non-relationship or direct fields.
Extensibility Framework Helpers	For detailed Extensibility Framework Helpers information, refer to the Helper Functions for TurboEngines Callbacks topic.

Best Practices for TurboEngines Callback code

Consider the following practices while writing the TurboEngines compatible C# callback code.

Whenever you change specific fields within the callback code and send them to the TurboPricing for pricing, it only verifies the change in Net Price and does not verify each field to detect the difference. Hence, if you make any change in callback code that impacts pricing, refer to the following.

TurboEngine uses a digest mechanism to detect if there are any changes to the line items. At the end of every pricing cycle, it recomputes the digest and compares it against the old digest. If they did not match, you could consider that the line item is changed. However, TurboEngines uses only a few selected fields (including NetPrice) to compute the digest but no custom field. Therefore, when the callback code makes any important change on a line item or custom field that TurboEngines might not detect, you must reset digest for that line item to avoid matching the new one. For example, use the following code to reset the digest for a line item with type *ILineItemModel*:

```
lineItem.Set("Digest", string.Empty);
```

Follow this simple pattern:

```
List<LineItem> lineItemsToResetDigest = new List<LineItem>();
foreach(var lineItem in allLineItems){
  // business logic that changes line item's field
  // add all such lines to above list
  lineItemsToResetDigest.Add(lineItem);
}

// reset the digest at the end in bulk
ResetDigest(lineItemsToResetDigest);
```

General Practices

1. Use class-level variables only when it is necessary and try to use the local variables as much as possible.
2. If you need to have a class-level variable, use *BeforePricingBatchAsync* (Base price callback) or *BeforePricingCartAdjustmentAsync* (Totalling callback) methods to initialize such class level variables.
3. Avoid using nested for loops.
4. Do not use **Try** or **Catch** in the callback code to explicitly log any unhandled exception. By default, an unhandled exception is logged by the TurboEngine Extensibility framework. You can also configure to display the unhandled exception on UI using the "Is Callback Exception Throw Enabled" (is-callback-exception-throw-enabled) feature flag.
5. For using Extensibility framework helpers (such as DBHelper, LogHelper, and so on) in your callback project, you must create a helper object in the callback implementation class (the main class of callback project which is implementing the callback interface). If you want to access helpers in any other supported class of your callback project, then you must pass the helper instances from the main implementation class to such classes.
6. PricingHelper's UpdatePrice() method on line items must not be used unless it is absolutely required. Callback implementation must be designed in such a way that, such a method call can be avoided.
7. Do not use concrete objects to set as a field value on the line item. In case if you want to set a concrete object as a value to the line items field, then stringify it before setting it on the line item field.

8. From TurboEngine callback implementation, do not make an SFDC API call using HttpHelper in any scenario. If such a scenario needs to be handled, get in touch with TurboEngine's product or technical team.
9. In Base price callback, always use batch line items for updating any value on line item as per your custom logic. Make sure that you do not update anything on cart line items when you are in the Base price callback.
10. Always use the async or await pattern for each method.
11. To access/update any field in callback code, the field API name/object API name must be spelled the same as it is in Salesforce.
12. Keep your code formatted for easy readability. Callback authoring UI provides code formatting capability to format the code as per C# standards.
13. Keep your code to have 0 or minimum code warnings. Callback authoring UI provides the capability that shows you the code warnings per the C# standards for your callback project. Ensure that you have fixed them and incorporated them in your code to have 0 or minimum code warnings.

Pricing Base Price Callback Interface

The *IPricingBasePriceCallback* interface provides you a mechanism to define a custom logic to be executed before, during, and after **Base Price** calculation. The *IPricingBasePriceCallback* is executed in batches which consists of the batch line items. TurboPricing calls this interface for every batch.

The following methods are available in the *IPricingBasePriceCallback* interface:

Method Signature	Description
Task BeforePricingBatchAsync(IBatchPriceRequest batchPriceRequest)	You can use this method to define custom logic that must be executed before Base Price is calculated.
Task OnPricingBatchAsync(IBatchPriceRequest batchPriceRequest)	You can use this method to define custom logic that must be executed during the Base Price calculation. You can use the price list items to write the custom logic.
Task AfterPricingBatchAsync(IBatchPriceRequest batchPriceRequest)	You can use this method to define custom logic that must be executed after the Base Price is calculated.

Method Signature	Description
Task OnProductOptionPriceAsync(IBatchPriceRequest batchPriceRequest, IDictionary<string, IProductOptionPrice> productOptionPrice)	You can use this extension point to modify product option prices for the given line item.
Extension Points The following extension points are added to the default template.	
Method Signature	**Description**

Method Signature	Description

Task OnPriceMatrixAsync(IBatchPriceRequest batchPriceRequest, IDictionary<string, IEnumerable<IPriceMatrixEntry>> priceMatrixEntries)	You can use this extension point to modify resolved price matrix for the given line item if required.
	
	The below code snippet is for reference purposes only.
	```
	/// <summary>
	/// On Price Matrix
	/// </summary>
	/// <param name="batchPriceRequest"></param>
	/// <param name="priceMatrixEntries"></param>
	/// <returns></returns>
	public async Task
	OnPriceMatrixAsync(IBatchPriceRequest
	batchPriceRequest, IDictionary<string,
	IEnumerable<IPriceMatrixEntry>>
	priceMatrixEntries)
	{
	    var batchLineItems =
	batchPriceRequest.GetLineItems().SelectMany(x
	=> x.GetChargeLines()).Select(s => new
	 LineItem(s)).ToList();
	    var cartLineItems =
	batchPriceRequest.GetCartContext().GetLineIte
	ms().SelectMany(x =>
	x.GetChargeLines()).Select(s => new
	 LineItem(s)).ToList();

	    var batchLineItem = batchLineItems[0];
	    if(priceMatrixEntries.ContainsKey(batchLi
	neItem.Id))
	    {
	        var priceMatrixEntriesEnumerable =
	priceMatrixEntries[batchLineItem.Id];
	        foreach(var pMatrixEntry in
	priceMatrixEntriesEnumerable)
	        {
	            pMatrixEntry.AdjustmentAmount =
	20;
	        }
	    }
	```

Method Signature	Description
	```
	    await Task.CompletedTask;
	}
	```

Method Signature	Description

Task OnPriceRuleAsync(IBatchPriceRequest batchPriceRequest, IDictionary<string, IEnumerable<IPriceRuleEntry>> priceRuleEntries)	You can use this extension point to modify the resolved price rule entries for the given line item if required.
	
	The below code snippet is for reference purposes only.
	
	`/// <summary>`
	`/// On PriceRule`
	`/// </summary>`
	`/// <param name="batchPriceRequest"></param>`
	`/// <param name="priceRuleEntries"></param>`
	`/// <returns></returns>`
	`public async Task OnPriceRuleAsync(IBatchPriceRequest batchPriceRequest, IDictionary<string, IEnumerable<IPriceRuleEntry>> priceRuleEntries)`
	`{`
	`    var batchLineItems = batchPriceRequest.GetLineItems().SelectMany(x => x.GetChargeLines()).Select(s => new LineItem(s)).ToList();`
	`    var cartLineItems = batchPriceRequest.GetCartContext().GetLineItems().SelectMany(x => x.GetChargeLines()).Select(s => new LineItem(s)).ToList();`
	
	`    var batchLineItem = batchLineItems[0];`
	`    if(priceRuleEntries.ContainsKey(batchLineItem.Id))`
	`    {`
	`        var priceRuleEntriesEnumerable = priceRuleEntries[batchLineItem.Id];`
	`        foreach(var pRuleEntry in priceRuleEntriesEnumerable)`
	`        {`
	`            pRuleEntry.AdjustmentAmount = 20;`
	`        }`
	`    }`
	`  await Task.CompletedTask;`

Method Signature	Description
	```
	}
	```

Method Signature	Description

Task OnPipelinePriceRuleAsync(IBatchPriceRequest batchPriceRequest, IDictionary<string, IEnumerable<IPriceRuleEntry>> pipelinePriceRuleEntries)	You can use this extension point to modify the resolved price pipeline rule entries for the given line item if required
	
	The below code snippet is for reference purposes only.
	
	`/// <summary>`
	`/// On Pipeline PriceRule`
	`/// </summary>`
	`/// <param name="batchPriceRequest"></param>`
	`/// <param name="pipelinePriceRuleEntries"></param>`
	`/// <returns></returns>`
	`public async Task OnPipelinePriceRuleAsync(IBatchPriceRequest batchPriceRequest, IDictionary<string, IEnumerable<IPriceRuleEntry>> pipelinePriceRuleEntries)`
	`{`
	`    var batchLineItems = batchPriceRequest.GetLineItems().SelectMany(x => x.GetChargeLines()).Select(s => new LineItem(s)).ToList();`
	`    var cartLineItems = batchPriceRequest.GetCartContext().GetLineItems().SelectMany(x => x.GetChargeLines()).Select(s => new LineItem(s)).ToList();`
	
	`    var batchLineItem = batchLineItems[0];`
	`    if(pipelinePriceRuleEntries.ContainsKey(batchLineItem.Id))`
	`    {`
	`        var pipelinePriceRuleEntriesEnumerable = pipelinePriceRuleEntries[batchLineItem.Id];`
	`        foreach(var ppRuleEntry in pipelinePriceRuleEntriesEnumerable)`
	`        {`
	`ppRuleEntry.AdjustmentAmount = 20;`

Method Signature	Description
	```
	            }
	        }
	        await Task.CompletedTask;
	    }
	```
Task OnPriceEscalatorAsync(IBatchPriceRequest batchPriceRequest, List<IPriceEscalator> priceEscalators)	You can use this extension point to modify the resolved price escalators if required.

Example Code

The below code snippet is for reference purposes only.

```
namespace Apttus.Lightsaber.Customer.Pricing
{
    public class PricingBasePriceCallback : CodeExtensibility,
IPricingBasePriceCallback
    {
        public async Task AfterPricingBatchAsync(IBatchPriceRequest
batchPriceRequest)
        {
            await Task.CompletedTask;
        }

        public async Task BeforePricingBatchAsync(IBatchPriceRequest
batchPriceRequest)
        {
            var batchLineItems = batchPriceRequest.GetLineItems().SelectMany(x =>
x.GetChargeLines()).Select(s => new LineItem(s)).ToList();

            //Example, setting custom field on line item before PLI resolution
            foreach(var batchLineItem in batchLineItems) {
                decimal extendedQuantity = batchLineItem.GetQuantity();
                decimal quantity = batchLineItem.GetQuantity();

                if(batchLineItem.IsOptionLine()) {
                    LineItem rootBundleLineItemModel = new
 LineItem(batchLineItem.GetRootParentLineItem().GetPrimaryLineItem());
```

```
                    decimal bundleQuantity = rootBundleLineItemModel.GetQuantity();
                    extendedQuantity = bundleQuantity * batchLineItem.GetQuantity();
                }

                batchLineItem.APTS_Extended_Quantity__c = extendedQuantity;
            }

            //You can also query DB here, and perform initial setup such as creating
required Dictionary, List and so on for later use in the callback code.

            await Task.CompletedTask;
        }

        public async Task OnPricingBatchAsync(IBatchPriceRequest batchPriceRequest)
        {
            var batchLineItems = batchPriceRequest.GetLineItems().SelectMany(x =>
x.GetChargeLines()).Select(s => new LineItem(s)).ToList();

            foreach(var batchLineItem in batchLineItems) {
                IPriceListItemModel priceListItemModel =
batchLineItem.GetPriceListItem();
                IPriceListItem priceListItemEntity = priceListItemModel.GetEntity();

                if(batchLineItem.PriceListId != priceListItemEntity.PriceListId) {
                       batchLineItem.APTS_Is_Contract_Pricing__c = true;
                }
            }
            await Task.CompletedTask;
        }
    }
}
```

Pricing Totalling Callback Interface

The *IPricingTotallingCallback* interface provides you a mechanism to defined custom logic to be executed before, during, and after adjustment calculation.
 The *IPricingTotallingCallback* is invoked once for each pricing request to calculate the total.

The following methods are available in the *IPricingTotallingCallback* interface:

Method Signature	Description
Task BeforePricingCartAdjustmentAsync(IAggregateCartRequest aggregateCartRequest)	You can use this method to define custom logic that must be executed before the adjustment is calculated.
Task AfterPricingCartAdjustmentAsync(IAggregateCartRequest aggregateCartRequest)	You can use this method to define custom logic that must be executed after the adjustment is calculated.
Task OnCartPricingCompleteAsync(IAggregateCartRequest aggregateCartRequest)	You can use this method to define custom logic that must be executed after the pricing is calculated completely.

Example Code

The below code snippet is for reference purposes only.

```
namespace Apttus.Lightsaber.Customer.Totaling
{
    public class PricingTotallingCallback : CodeExtensibility,
IPricingTotallingCallback
    {
        public async Task BeforePricingCartAdjustmentAsync(IAggregateCartRequest
aggregateCartRequest)
        {
            var cartLineItems =
aggregateCartRequest.GetCartContext().GetLineItems().SelectMany(x =>
x.GetChargeLines()).Select(s => new LineItem(s)).ToList();

            foreach(LineItem cartLineItem in cartLineItems) {
                if(cartLineItem.IncentiveBasePrice.HasValue &&
cartLineItem.IncentiveBasePrice.Value != 0) {

                    decimal sellingTerm =
cartLineItem.GetValuetOrDefault(LineItemPropertyNames.SellingTerm, 1);
                    decimal lineItemQ = cartLineItem.GetQuantity();

                    decimal? unitIncentiveAmount = cartLineItem.BasePrice -
cartLineItem.IncentiveBasePrice;
```

```
                    cartLineItem.APTS_Unit_Incentive_Adjustment_Amount__c =
unitIncentiveAmount;
                    cartLineItem.IncentiveBasePrice = cartLineItem.BasePrice -
unitIncentiveAmount;
                    cartLineItem.IncentiveAdjustmentAmount = unitIncentiveAmount *
lineItemQ * sellingTerm * -1;
                }
            }

            await Task.CompletedTask;
        }

        public async Task AfterPricingCartAdjustmentAsync(IAggregateCartRequest
aggregateCartRequest)
        {
            //Example, Set custom fields on line item based on the adjusted price
            await Task.CompletedTask;
        }

        public async Task OnCartPricingCompleteAsync(IAggregateCartRequest
aggregateCartRequest)
        {
            var cartLineItems =
aggregateCartRequest.GetCartContext().GetLineItems().SelectMany(x =>
x.GetChargeLines()).Select(s => new LineItem(s)).ToList();
            foreach(var cartLineItem in cartLineItems) {
                if(cartLineItem.NetPrice < 1000) {
                    cartLineItem.APTS_Deal_Color = "Red";
                } else {
                    cartLineItem.APTS_Deal_Color = "Green";
                }
            }
            await Task.CompletedTask;
        }
    }
}
```

Related Pricing Callback Interface

The interface provides you the capability to define custom logic for calculating the pricing for related product line items. The prices in the related product line items are dependant on the price of other line items.

The following method is available in the *IRelatedPricingCallback* interface:

Method Signature	Description
Task<List<IRelatedPricingBatchResponse>> ComputeBasePriceBatchAsync(IRelatedPricingBatchRequest relatedPricingBatchRequest);	You can use this method to define custom logic that must be executed to specify a new base price for the related line item.

Example Code

The below code snippet is for reference purposes only.

```
namespace Apttus.Lightsaber.Customer.RelatedPricing
{
    public class RelatedPricingCallback : CodeExtensibility, IRelatedPricingCallback
    {
        public async Task<List<IRelatedPricingBatchResponse>>
ComputeBasePriceBatchAsync(IRelatedPricingBatchRequest relatedPricingBatchRequest)
        {
            List<IRelatedPricingBatchResponse> relatedPricingBatchResponseResult =
new List<IRelatedPricingBatchResponse>();

            List<ILineItemModel> relatedLineItems =
relatedPricingBatchRequest.GetRelatedLineItems();

            foreach(var relatedLineItem in relatedLineItems) {

                var priceBreakupRecords = relatedLineItem.GetPriceBreakupRecords();

                foreach(var priceBreakup in priceBreakupRecords) {
                    if(priceBreakup.BreakupType == "Total" &&
relatedLineItem.GetEntity().RelatedAdjustmentAmount > 5000) {
                        IRelatedPricingBatchResponse relatedPricingResponse =
relatedPricingBatchRequest.CreateRelatedPricingBatchResponse(relatedLineItem);
                        relatedPricingResponse.BasePrice = 500;

relatedPricingBatchResponseResult.Add(relatedPricingResponse);
                    }
                }
            }
            return await Task.FromResult(relatedPricingBatchResponseResult);
        }
    }
}
```

Migrating Salesforce Pricing Callback to TurboEngines Pricing Callback

This section provides high-level details on mapping the pricing callback from Salesforce to TurboEngines.

Mapping pricing callback from Salesforce to TurboEngines

This section summarizes the mapping process from the Salesforce pricing callback method to TurboEngines pricing callback methods. The 'Mode' plays an important role in the mapping process as the mode Salesforce pricing callbacks are now available in dedicated interfaces in TurboEngine pricing callback. *IPricingBasePriceCallback* for BASEPRICE mode and *IPricingTotallingCallback* for ADJUSTMENT mode.

The following table summarizes the **BASEPRICE** mode methods of Salesforce pricing callback.

BASEPRICE Mode in Salesforce	IPricingBasePriceCallback in TurboEngine
void start(ProductConfiguration cart);	Task BeforePricingBatchAsync(IBatchPriceRequest batchPriceRequest)
void setMode(PricingMode mode);	Not Applicable
void beforePricing(ProductConfiguration.LineItemColl itemColl);	Task BeforePricingBatchAsync(IBatchPriceRequest batchPriceRequest)
void onPriceItemSet(PriceListItem__c itemSO, LineItem lineItemMO);	Task OnPricingBatchAsync(IBatchPriceRequest batchPriceRequest)
void afterPricing(ProductConfiguration.LineItemColl itemColl);	Task AfterPricingBatchAsync(IBatchPriceRequest batchPriceRequest)
void finish();	Task AfterPricingBatchAsync(IBatchPriceRequest batchPriceRequest)
	Task OnProductOptionPriceAsync(IBatchPriceRequest batchPriceRequest, IDictionary<string, IProductOptionPrice> productOptionPrice)
	Task OnPriceMatrixAsync(IBatchPriceRequest batchPriceRequest, IDictionary<string, IEnumerable<IPriceMatrixEntry>> priceMatrixEntries)

BASEPRICE Mode in Salesforce	IPricingBasePriceCallback in TurboEngine
	Task OnPriceRuleAsync(IBatchPriceRequest batchPriceRequest, IDictionary<string, IEnumerable<IPriceRuleEntry>> priceRuleEntries)
	Task OnPipelinePriceRuleAsync(IBatchPriceRequest batchPriceRequest, IDictionary<string, IEnumerable<IPriceRuleEntry>> pipelinePriceRuleEntries)
	Task OnPriceEscalatorAsync(IBatchPriceRequest batchPriceRequest, List<IPriceEscalator> priceEscalators)

The following table summarizes the **ADJUSTMENT** mode methods of Salesforce pricing callback.

ADJUSTMENT Mode in Salesforce	IPricingTotallingCallback in TurboEngine
void start(ProductConfiguration cart);	Task BeforePricingCartAdjustmentAsync(IAggregateCartRequest aggregateCartRequest)
void setMode(PricingMode mode);	Not applicable
void beforePricing(ProductConfiguration.LineItemColl itemColl);	Task BeforePricingCartAdjustmentAsync(IAggregateCartRequest aggregateCartRequest)
void beforePricingLineItem(ProductConfiguration.LineItemColl itemColl, LineItem lineItemMO);	Task BeforePricingCartAdjustmentAsync(IAggregateCartRequest aggregateCartRequest)
void afterPricingLineItem(ProductConfiguration.LineItemColl itemColl, LineItem lineItemMO);	Task AfterPricingCartAdjustmentAsync(IAggregateCartRequest aggregateCartRequest)

ADJUSTMENT Mode in Salesforce	IPricingTotallingCallback in TurboEngine
void afterPricing(ProductConfiguration.LineItemColl itemColl);	Task AfterPricingCartAdjustmentAsync(IAggregateCartRequest aggregateCartRequest)
void finish();	Task OnCartPricingCompleteAsync(IAggregateCartRequest aggregateCartRequest)

- The mapping shown above is for reference only and varies based on the use case implemented in the Salesforce pricing callback.
- Review the Salesforce pricing callback code and put functional instances in equivalent TurboEngine extension points.

Use Case

For example, in the Salesforce pricing callback's *BeforePricing* method, there is a **logic** set to a custom field value on a line item based on LineItem's resolved PLI's list price. The **logic** is not included any mode (BASEPRICE or ADJUSTMENT) check.

The migration process for the given SFDC pricing callback code to the TurboEngine callback code and where to insert the code in the TurboEngine callback code is explained below.

From a functional point of view, the logic is applicable for ADJUSTMENT mode's *BeforePricing* method. Because, in BASEPRICE mode's *BasePricing* method, LineItem's resolved PLI is not available. Hence, the most probable mapping point in TurboEngines is the BeforePricingCartAdjustmentAsync method of IPricingTotallingCallback. On the other hand, since the logic uses just resolved PLI's list price, another probable mapping point in TurboEngines is the *OnPricingBatchAsync* method *IPricingBasePriceCallback* where line item’s resolved PLI details are available.

Onboarding Data Sync Services

To complete post-provisioning for TurboConfig & TurboPricing, the tenant admin must configure settings for data sync services. TurboEngines data sync provides a high-performance mechanism to sync config and pricing master data at regular, scheduled intervals (or on-demand) between Conga CPQ on Salesforce, TurboPricing, and TurboConfig. Before the initial data sync, you must configure settings enable data sync services and give the administrator access to the TurboEngines Data Sync Admin user interface (UI) to set up and schedule or activate the sync.

Perform the following tasks to complete post-provisioning data sync tasks for TurboConfig and TurboPricing.

Configuring Data Sync Specific Settings

You must configure the data sync service URL and a CSP Trusted Site entry so SFDC can communicate with an external server.

To configure the service URL

1. Go to **Setup > Custom Settings**.
2. Go to **Turbo Engine Admin Properties**.
3. Click **Manage**.
4. Click **New**.
5. Enter the following required properties:
    - Name: *LightsaberServiceUrl*.
    - TurboEngines service Endpoint – Endpoint URL provided by Conga CloudOps
6. Click **Save**.

ⓘ Do not enter the forward-slash (/) symbol at the end of the Endpoint URL.

To configure the CSP Trusted Site

1. Go to **Setup > CSP Trusted Sites**
2. Click **New Trusted Site**.
3. Enter the following required properties:
    a. Enter a **Trusted Site Name** for the trusted site (for example, "TurboEngineAdminService")
    b. Enter the **Trusted Site URL** (this is the service URL from the previous task).
4. Click **Save**.

Enabling My Domain

As an additional prerequisite to using the Data Sync Admin, you must deploy the "My Domain" feature in Salesforce.

For more information, refer to https://developer.salesforce.com/docs/atlas.en-us.lightning.meta/lightning/intro_reqs_my_domain.htm and https://help.salesforce.com/articleView?id=domain_name_overview.htm&type=5

Configure Permissions for Data Sync Admin User

Users who need to configure and run TurboEngines Data Sync must have permission to access and use the Data Sync Admin UI. This can be a user assigned to the System Administrator profile, or you can customize a profile and create one or more users in this role.

To check if the current user has the right permissions:

1. Log in to your organization as the admin user.
2. Open the Salesforce App Launcher (Lightning) and launch the **TurboEngines Admin** app.
3. If the **Data Integration** and **Callbacks** tabs are visible after launching the app, the user has the correct permissions. Otherwise, log back in as a system administrator and perform the following tasks to provide access to the user profile.

To provide access to the data sync app

1. Go to **Setup > App Manager**.
2. Find the **TurboEngines Admin** app in the list. Click the drop-down and the end of the row and select **Edit**.
3. Click **User Profiles**.
4. From the list of Available Profiles, search and select the app you want to add.
5. Click the right-facing arrow to move the profile from the list of Available Profiles to the list of Selected Profiles.
6. Click **Save**.

To make all tabs visible in the data sync app

1. Go to **Setup > Profiles**.
2. Search for the profile you want to configure and click **Edit**.
3. Under Custom Settings, make sure the following tabs are set as "Default On":
    - **Data Integration**: This tab serves as the starting point for managing all consumer profiles.
    - **Callbacks**: This tab allows you to manage pricing callbacks for TurboPricing.
    - **Consumer Profile**: This tab allows you to set up and configure data sync operations.
    - **Run Details**: This tab allows you to review run history for data sync and take action.

Configuring Single Currency Orgs

Exchange rates are required for currency conversion during pricing and therefore must be synced by TurboEngines. Since single currency orgs do not have a currencies table, any attempt to sync pricing data will fail. Take one of the following two actions when onboarding the org prior to the initial sync:

- Enable multiple currency in the provisioned org (see Salesforce documentation).
- Raise a support ticket to have currencies removed from the Conversion Rates and Custom Settings consumer profile.

Salesforce orgs are provisioned with a single currency. In such cases, the currencies table will not exist. Only when the administrator enables multiple currency will table be created and hold the exchange rates.

Setting up and Syncing TurboConfig Data

For complete information and the tasks required to administer TurboEngines data sync for TurboConfig and TurboPricing, refer to *Data Sync Administrator Guide.*

Frequently Asked Questions (TurboConfig)

What is TurboConfig and how does it work?

TurboConfig is a configuration engine to process product configuration rules while configuring products and finalizing a quote. TurboConfig offloads the computation workload from the Salesforce platform to the Conga Flexible Compute Platform built using microservices. The benefit of the TurboConfig is that users can sell complex configurations much faster because of significantly-optimized processing time. Also, it allows customers to expand the solution to other business units and sell faster.

For example, in a TurboConfig-enabled flow, when the Sales Representative adds a product or a favorite configuration to the cart, the application of complex constraint rules associated with them is offloaded to the Conga Flexible Compute Platform to process for faster response.

When do I need to use TurboConfig?

The Salesforce platform has limitations (such as heap size, CPU timeout limits, number of SOQL limits, and view state) that result in slower response times and usability issues. TurboConfig handles such complex rules and processes a volume of rules significantly faster.

TurboConfig is recommended when you have a large number of rules or highly complex configuration rules to be applied while selecting a product or configuring a bundle.

For example

- If you have more than 100 constraint rules (inclusion, exclusion, recommendation, and replacement rules) applicable across standalone and bundle products
- If you have more than 50 field expressions applicable across products and bundles
- If you have more than 100 product attribute value rules applicable across bundles
- If you have a complex bundle structure that includes more than 500 options and several option groups
- If you have complex bundles rules such as min/max, custom filter callback, repeat inclusions.

How do I enable TurboConfig?

You must have a license for TurboPricing or TurboConfig to enable either service. If you do not have a license, please contact your Conga Account Executive before you begin. After you acquire a license TurboConfig instances will be provisioned for you.

For detailed instructions on how to enable TurboConfig, refer to instructions on how to turn on TurboConfig.

Which version of CPQ should I be on to use TurboConfig?

You must be on CPQ on Salesforce Summer 2020 release or above.

What are the supported features in TurboConfig?

For a complete list of supported features on TurboConfig, refer to the feature matrix.

Is the TurboConfig available for all products or only select products?

You can use TurboConfig for all or select products. However, the TurboEngines data sync services configure the selected products at a regular frequency.

For instructions on how to onboard the data sync services to TurboConfig, refer to the instructions here.

How does TurboConfig work with Data Sync for synchronizing the data?

TurboEngines data sync provides a high-performance mechanism to sync the config and pricing master data at regular, scheduled intervals (or on-demand) between Conga CPQ on Salesforce, Turbo Pricing, and Turbo Config. To start using the data sync services, the tenant admin must configure settings for data sync services. For more information on configuring data sync settings, refer to Onboarding data sync services.

Is there a way to automate the data sync of all products and changes on a regular basis?

Yes, TurboConfig administrators can now make use of the TurboEngine Data Sync Admin application to configure and manage master data sync at regular scheduled intervals and on-demand.

Does TurboConfig work on existing quote or configurations, which were created using a different constraint rule execution mode?

Yes, if you have quotes in progress and if you have configured quotes using the *Client* execution mode, you can process the quotes using the *CMS* execution mode. However, if you have created a quote using the *CMS* execution mode when you switch to the *Client* execution mode, you may have to delete the line items and add them again.

Can I configure custom flows for TurboConfig?

Yes, You can enable TurboConfig for selective CPQ flows. You can use this functionality to avoid making TurboConfig as the default configuration engine and use the engine to process large and complex configuration rules.

For more information on configuring custom flows, refer to Configuring Custom Flows for TurboConfig.

How do I switch from Server Side/Client Side constraint rules to TurboConfig?

For detailed instructions on how to enable TurboConfig, refer to instructions on how to turn on. Also, refer to the feature parity matrix before you switch to TurboConfig. Note that custom callbacks are not supported in this release.

When I refresh my Salesforce Sandbox org, should I change any config settings?

When you refresh your sandbox, you must reconfigure TurboConfig after the refresh. Follow the onboarding process to enable TurboConfig in your sandbox after you have refreshed the sandbox.

I have done some customization in my org such as added formula fields, workflow rules, callbacks. Do my customizations work when I switch to TurboConfig?

There is no impact on any customizations you may have done on CPQ Objects. However, if you have written any configuration callbacks such as Option Filter Callbacks, you will be required to migrate your callback to TurboConfig using the microservice callback framework. Note that the callbacks are not supported in TurboConfig in the Winter 2020 release. Refer to the supported feature matrix before switching to TurboConfig engine.

Is TurboConfig supported to work with ABO and Service CPQ?

No. ABO and Service CPQ are not supported to work with TurboConfig in the Winter '20 release. For a complete list of supported features on TurboConfig, refer to the Feature Matrix.

Is TurboConfig security and privacy compliant?

TurboEngines run in a secure multi-tenant environment and TurboEngines are designed to provide full security and privacy with your data. The services are hosted in IBM Cloud, which is ISO 27001/2, SOC 1/2, GDPR compliant. Conga takes advantage of data encryption and access control features enabled by the cloud service provider. If you have any questions or need details, contact Conga Technical Support.

TurboEngines Data Sync Documentation

Select one of the following topics for more information:

- About TurboEngines Data Sync
- What's New in Data Sync Documentation
- TurboEngines Data Sync for Administrators

About TurboEngines Data Sync

TurboEngines Data Sync service is a critical component of the **Conga TurboEngines** platform that provides a high-performance mechanism to sync master data at regular, scheduled intervals (or on-demand) between Conga CPQ on Salesforce and specific consumer endpoints using Conga's **Flexible Computing Platform**.

About TurboEngines

Conga TurboEngines is a concurrent processing engine provided by Conga that comprises various microservices that process product rules and configurations, pricing configurations and data, and other product-related business data. TurboEngine offloads the computation workload from the Salesforce platform to the Conga Flexible Compute Platform to reduce the processing time on the cart. Processing the computation workload in the Conga Flexible Compute Platform reduces the interaction costs and the quote turnaround time specifically during peak load or large transactions.

TurboEngines scales on 3 dimensions:

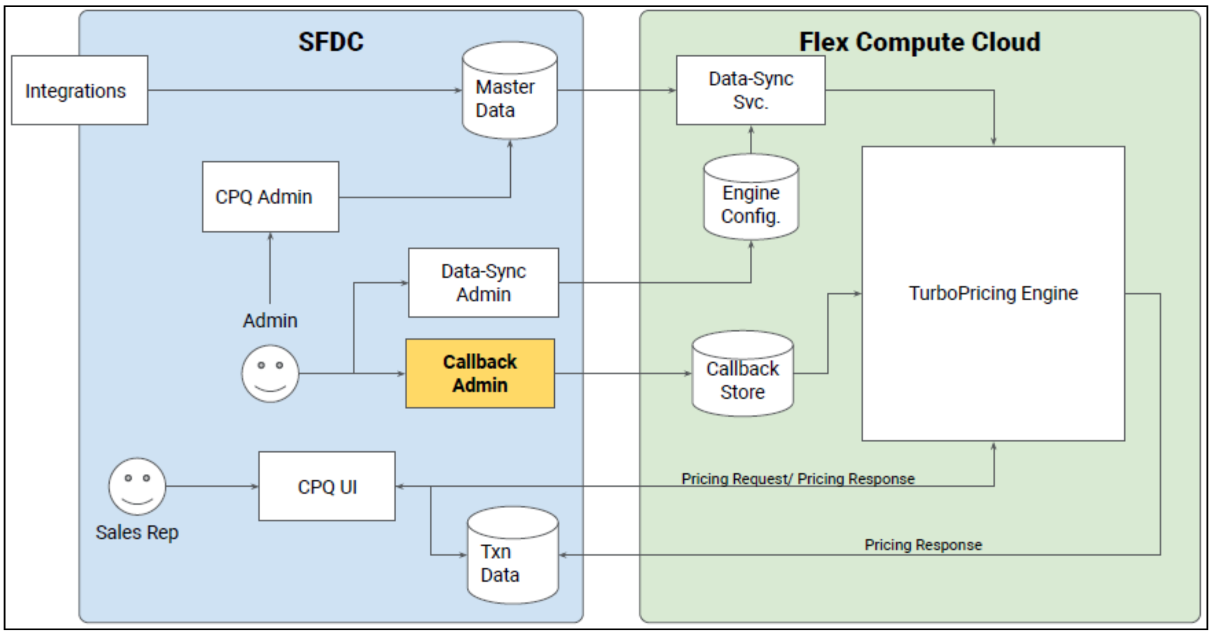
- Number of users
- Size of transaction
- The complexity of product and pricing configuration

For more information on how to get started with and configure **TurboEngines**, please refer to the *Conga TurboEngines Spring '21 Administrator Guide* on the Conga documentation portal.

About the TurboEngine Data Sync Flow

TurboEngines Data Sync services comprise several components that work to pull data from Conga applications in Salesforce to a staging database that is then delivered using consumer services to various consumer endpoints (Consumer Profiles) based on their need for data. The data sync service pulls data from Salesforce at regular, scheduled intervals (using scheduler) and as needed based on Salesforce Push Topic configuration. A user can also retrieve data by invoking on demand sync from the Data Sync Admin UI.

Refer to the following diagram for a high-level flow of the data sync process between SFDC and TurboEngines.



About Consumer Profiles

A Consumer Profile as defined in Data Sync is essentially a master list and format definition for objects, fields, and the related objects and fields to be synced to a specific consumer endpoint. In the Data Sync Admin user interface, the Consumer Profile is defined as **Sync Settings**, comprising object and field data to be synced, details or indicators related to the profile, and sync frequency settings.

For more information on Consumer Profiles, refer to Navigating the Data Sync Admin User Interface.

Data Sync Prerequisites

Provisioning requirements for Conga TurboEngines must be met prior to managing data sync. Refer to Enabling TurboEngines in an Org in the *Conga TurboEngines Winter '20 Administrator Guide* for more information.

Key Terms

Term	Definition
TurboEngines	A concurrent processing engine provided by Conga that comprises various microservices that process and sync product configurations, pricing configurations, and data.
Flexible Computing Platform	A Conga-designed cloud platform built using microservices that offloads the computation workload from the Salesforce platform to reduce processing time on the cart, interaction costs, and the quote turnaround time specifically during peak load or large transactions.
TurboPricing	A pricing engine microservice on the Flexible Computing Platform that computes complex pricing computations and callbacks.
TurboConfig	A configuration engine microservice on the Flexible Computing Platform that computes complex product configurations and product rules.
Data Sync	The process handled by TurboEngines Data Sync Services to sync master pricing data at regular, scheduled intervals or on-demand.
Run History	A log of all data successful and unsuccessful data sync executions provides a means of troubleshooting sync operations using helpful error messages.
Consumer Profile	A master list and format definition for objects, fields, and related objects and fields to be synced to various consumer endpoints.
Consumer Service	The service that delivers synced data to the consumer endpoint.
Consumer Endpoint	The destination for synced data delivered by a Consumer Service and defined by the Consumer Profile.

What's New in Data Sync Documentation

The following table lists changes in documentation to support each release.

Document	Publication Date	Topic	Description
Summer '21	06 Jul 2021	Adding Objects and Fields for Sync	Modified topic based on changes to the Update Fields flows and changes in the Sync Settings UI.
		Creating Sync Indexes	Modified topic to add details for the sync index sequence.
		Working with Data Sync Run History	Modified topic to add details for the admin icon, resync tracking, new link (provides additional pricing master data sync details), in the Run History UI.
		Viewing and Evaluating Error Messages	Modified topic based on changes to downloading error messages in the Run History UI.
Spring '21 (Rev. A)		Adding Objects and Fields for Sync	Updated topic based on changes to the Manage Fields flow in the Sync Settings UI.
		Creating Sync Indexes	Updated topic based on changes to restrictions on Sync Indexes.
		Working with Data Sync Run History	Updated topic to add details for the Resync Sync Type.
		Data Sync Limitations	Updated topic to remove the limitation regarding syncing of future fields since the feature was removed.
Spring '21		NA	No updates.
Winter '20		Winter '20 Data Sync Administrator Guide	Updated topic to add a minor note about TurboConfig.
		About TurboEngines Data Sync	Updated topic to add a definition for TurboConfig.

Document	Publication Date	Topic	Description
		Navigating the Data Sync Admin User Interface	Updated topic to add consumer profile information for TurboConfig.
		Adding Objects and Fields for Sync	Updated topic to reflect changes to managing fields and related fields.
		Managing Email Notifications	New topic.
		Running Data Sync On-Demand	Updated topic to add a note about email notifications.
		Creating Sync Indexes	New topic.
		Enabling Objects for Push-Based Sync	New topic.
		Working with Data Sync Run History	Updated topic to add a note about email notifications and descriptions for new sync status values.
		Viewing and Evaluating Error Messages	Updated topic with new error messages.
		Retrying Data Sync	New topic.
		Data Sync Limitations	Updated topic name.
		Data Sync FAQ	New topic.
Summer 2020 (Rev. A)		Navigating the Data Sync Admin User Interface	Updated topic to reflect the name changed to the "TurboEngines Admin" app.

Document	Publication Date	Topic	Description
		Adding Objects and Fields for Sync	Updated topic to reflect minor changes to the simple and complex object tasks.
		Working with Data Sync Run History	Updated topic to include details for in-progress sync.
Summer 2020		All topics.	First release (internal only).

TurboEngines Data Sync for Administrators

This section provides Data Sync Administrators with the information required to manage master data sync for Conga Commerce implementations that have enabled **Conga TurboEngines** as the primary pricing and configuration engine.

Topic	Description
What's Covered	This guide provides information for administrators to manage Conga TurboEngines Data Sync.
Primary Audience	• Conga Administrators • Conga Professional Services • Pricing Administrators • Customer Administrators
IT Environment	Refer to Getting Started for System Requirements and Supported Platforms.
Updates	For a comprehensive list of updates to this guide for each release, see the What's New in Data Sync Documentation topic.
Other Resources	• *Conga TurboEngines Documentation* • *Conga TurboEngines Release Notes* • *Conga CPQ Documentation*

This guide describes the following tasks:

- Managing Data Sync Settings

    - enabling and disabling data sync
    - adding objects and fields
    - adding sync actions and formats
    - running data sync
    - scheduling data sync
- Working with Data Sync Run History
    - viewing and evaluating run history
    - viewing error messages
- Troubleshooting Data Sync (known issues)

Before using TurboEngines Data Sync, you must be familiar with the following:

- Basic Salesforce administration
- Salesforce Lightning experience
- Salesforce and Conga terms and definitions
- Basic understanding of Conga TurboEngines
- Basic understanding of Conga TurboPricing
- Basic understanding of Conga TurboConfig

DOC ID: CTESUM21DSAG20210707

Navigating the Data Sync Admin User Interface

The TurboEngines Data Sync administrator user interface allows administrators to manage, modify, and monitor data sync operations of master data between Conga on Salesforce and various consumer endpoints. Administrators can make changes to data sync consumer profiles by adding, updating, or deleting objects and fields for sync. They can also monitor data sync jobs status and run history, as well as manually trigger data sync for any given consumer profile.

Navigating to TurboEngines Data Sync

To open the Data Sync Admin UI, go to **Salesforce App Launcher > TurboEngines Admin**.

Go to the **Data Integration** tab. From the list of Consumers, click **Manage Sync** to open the Admin UI.

The **Callbacks** tab is used for managing TurboPricing Callbacks. Refer to the Conga TurboEngines Summer '21 Administrator Guide for information on setting up and managing callbacks.

About Consumer Profiles

The standard and custom objects that are synced from Salesforce to TurboEngines are defined as Consumer Profiles. Consumer Profiles are a master list and format definition for objects, fields, and the related objects and fields to be synced to a specific consumer endpoint. These profiles are displayed in the Consumer list under the Data Integration tab. Consumer Profiles are classified into three Consumer Groups based on the consuming application (TurboPricing or TurboConfig).

TurboPricing Consumer Group

Consumer Profile	Data Sync Admin Access	Description
Pricing Master Data	Read / Write	Seeded master pricing data (Salesforce standard and Conga custom objects). This consumer profile can be modified from Sync Settings. The following objects will not be synced during initial sync and will be marked as **Skipped** in the Run History Detail page: • Complex_Apttus_Config2__PriceRule__c • Complex_Apttus_Config2__PriceMatrix__c • Apttus_Config2__ProductGroup__c • Complex_Apttus_Config2__PriceRuleset__c • Complex_Apttus_Config2__RelatedPriceListItem__c • Product2 These objects are synced as part of other objects in the form of denormalization. If the installed CPQ version is less than 13.1909 in org, remove the following fields from the **Pricing Master Data** consumer profile. • **Apttus_Config2__PriceRuleset__c** > **Apttus_Config2__ParentPricePipelineId__c 2** • **Apttus_Config2__PriceRuleset__c** > **Apttus_Config2__StartPricePointId__c 3** • **Apttus_Config2__PriceRuleset__c** > **Apttus_Config2__EndPricePointId__c** • **Apttus_Config2__PriceRule__c** > **Apttus_Config2__PricePoints__c**
Conversion Rates and Custom Settings	Read-only	Seeded custom settings for pricing. This consumer profile is read-only.

Consumer Profile	Data Sync Admin Access	Description
Deal Maximizer Master Data Tables	Read-only	Seeded master pricing data for Conga Deal Maximizer. You can modify the consumer profile from Sync Settings (no new objects can be added).

TurboConfig Consumer Group

Consumer Profile	Data Sync Admin Access	Description
Configuration Master Data	Read / Write	Seeded master product configuration data (Salesforce standard and Conga custom objects). This consumer profile can be modified from Sync Settings.

Extensibility

Consumer Profile	Data Sync Admin Access	Description
Customer-Specific Data	Read / Write	Seeded consumer profile with no data. This consumer profile must be modified from Sync Settings to add customer-specific data.

Overview of the Data Sync Admin UI

The TurboEngines Admin UI comprises two main tabs:

- **Sync Settings**: Manage objects and fields to sync for the selected consumer profile. You can view objects, fields and their hierarchy and make modifications as necessary. You can also define the frequency at which the data will be synced.

- **Run History**: View data sync run history and associated error messages. The tab displays a list of data sync Ids and other information, including whether or not the sync was successful. You can click on an entry in the run history to view any error messages or other details that are provided.

Managing Sync Settings

Refer to the topics in this section for step-by-step information on enabling, configuring, and executing data sync from Sync Settings in the Admin UI.

- Enabling and Disabling Data Sync
- Adding Objects and Fields for Sync
- Creating Sync Indexes
- Running Data Sync On-Demand
- Managing Data Sync Frequency
- Managing Email Notifications

Enabling and Disabling Data Sync

You can enable or disable data sync for a consumer profile from the **Sync Settings** tab.

To enable or disable data sync for a consumer profile

1. From the list of Consumers, click **Manage Sync** to manage data sync for the profile you want to enable or disable.
2. From the Sync Settings header, click **Sync Enable** toggle to enable or disable data sync. A message is displayed notifying you that the data sync is enabled or disabled.

When data sync is disabled for a consumer profile:

- The **Sync Now** button is disabled.
- Any in-progress sync is not affected and completes as normal.
- Any scheduled synchronizations are cancelled (note that the sync frequency setting for this profile is still set in the event the sync is re-enabled).
- The admin can still modify objects and fields for sync.
- The admin can still modify sync frequency.

Adding Objects and Fields for Sync

The initial list of objects to be synced is provided by the associated consumer profile and displayed on the Sync Settings page. When you need to add new objects or fields to the sync or update the existing data structure, you can do so from Sync Settings. You have the option to add simple (single object with no joins) or complex objects (single object with one or more joins). When adding new objects or managing existing objects, you can also select which fields to include or exclude from the sync operation.

> ⓘ The Add Objects button is disabled for consumer profiles that are read-only.

To add a simple object to the sync

Add a simple object to the sync when you only need the object and its fields (included the Id of any reference fields).

1. Go to the **Sync Settings** page.
2. From the object list at the bottom of the page, click **Add Object**. The Add Object dialog is displayed.
3. Select **Simple Object**.
4. Click **Next**.
5. Search and select the object to sync.
6. Click **Next**.
7. Select **Data and metadata sync** to sync both object data and metadata, or select **Only metadata sync** to sync only metadata for the object.

   > ⓘ When you choose to sync only metadata, click **Save**. The sync profile is updated. No further action is required.

8. Click **Manage**. The Manage Object page is displayed.
9. Click **Manage Fields** to the right of the object name. The Manage Fields dialog is displayed.
10. Search and select one or more fields to add them to the sync. If you want to sync all fields for the selected object click checkbox to the left of Field Name in the table header.

    > ⓘ Formula fields are distinguished from non-formula fields by **[Data Type] (Formula Field)**.

11. Click **Submit**. The fields are added. Click the chevron (>) next to the object name to expand the view to include the fields you just added.
12. Click **Submit**. The object is added to the list of objects on the Sync Settings page.

To add a complex object to the sync

Add a complex object to the sync when you want to include joins for related objects and their fields (reference, child, and nested relationships).

1. Go to the **Sync Settings** page.
2. From the object list at the bottom of the page, click **Add Object**. The Add Object dialog is displayed.
3. Select **Complex Object**.
4. Click **Next**.
5. Search and select an object to sync.
6. Click **Next**.
7. Enter the **Target Object Name**. This can be any user-friendly name for the object (do not include spaces).
8. Click **Manage**. The Manage Object page is displayed for the object you selected. By default, only the top-level object is displayed in the list.
9. Click **Manage Fields** to the right of the object name. The Manage Fields dialog is displayed.
10. Search and select one or more fields to add them to the sync. To sync all fields for the selected object click the checkbox to the left of Field Name in the list header.

    Formula fields are distinguished from non-formula fields by **[Data Type] (Formula Field)**.

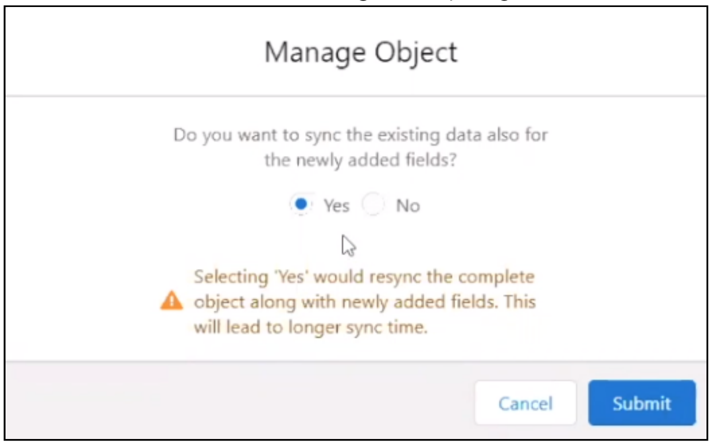
11. Click **Submit**. The Manage Objects page refreshes to display an expanded list of the object and fields to be synced.
12. To add reference or child objects, click **Manage Relations**. The Manage Relations dialog is displayed.
13. From the Reference Objects tab, search and select one or more lookup fields to add them to the sync. To sync all reference fields, click the checkbox to the left of Reference Object in the list header.
14. From the Child Objects tab, search and select one or more child objects to add them to the sync. To sync all child objects, click the checkbox to the left of Child Object in the list header.
15. Click **Submit** to add all selected reference and child objects.
16. Reference and child object fields can be added as joins to the nth level:

   a. To manage fields for a *reference* object you added, click the chevron (>) to the left of the reference field name and click **Manage Fields**. To manage additional reference and child objects, click **Manage Relations**.
   b. To manage fields for a *child* object you added click **Manage Fields** to the right of the object name. To manage additional reference and child objects, click **Manage Relations**.
17. Click **Submit**. Click the chevron (>) to the left of a field name to view all fields, reference objects, and child objects you added for that field.
18. Click **Submit**. The object and its selected fields is added to the list of objects on the Sync Settings page.

To update an object for sync

Update objects when you want to include or exclude fields from the sync.

1. Go to the **Sync Settings** page.
2. Search for the object you want to modify.
3. Click the drop-down (▾) at the end of the row and select **Manage Object**. The Manage Object page is displayed.
4. Click **Manage Fields** to open the Manage Fields dialog. Click the checkbox next to any field in the list to enable or disable the field for sync and click **Submit**.
5. Click **Submit**. A new dialog is displayed.


Manage Object

Do you want to sync the existing data also for the newly added fields?

[✓ Yes] No

Selecting 'Yes' would resync the complete object along with newly added fields. This will lead to longer sync time.

6. Under **Do you want to sync existing data also for the newly added fields?** select Yes or No:
   - Select **Yes** to sync existing data for the entire object included any newly added fields.
   - Select **No** to decline the sync. The next sync will be run depending on any changes to fields marked for sync or the next scheduled or manual sync.

> ⓘ Be careful when choosing to resync the entire object as it will lead to a longer sync time.

7. Click **Submit** to return to the Manage Object page.
8. For complex objects, click **Manage Relations** to open the Manage Relations dialog. Click the checkbox next to any reference field or child object to enable or disable the field for sync. Click **Submit** to return to the Manage Object page.
9. Click **Submit** to save your changes and return to the Sync Settings page.
10. You can now trigger On-demand Sync to sync your changes to fields marked for sync.

To delete an object

1. Go to the **Sync Settings** page.
2. Search for the object you want to delete.
3. Click the drop-down at the end of the row (▾) and select **Delete**. A confirmation dialog is displayed.

   > ⓘ You cannot delete seeded objects. You cannot delete fields marked with the lock (🔒) icon.

4. Click **Yes** to confirm or **No** to cancel.

   > ⓘ Be careful when deleting objects. To maintain sync integrity, delete child or reference objects first.

Enabling Objects for Push-Based Sync

You can enable a simple or complex object for Push-based sync. This takes advantage of PushTopic Events in Salesforce that notify the data sync service when one or more records have been created, updated, or deleted, or when changes have been made to a record based on a specific PushTopic query. When you enable an object for PushTopic sync, any time activity in Salesforce meets the criteria for a PushTopic Event, the corresponding object and its data will be synced for the applicable consumer profile.

To enable an object for PushTopic sync

1. Go to the Sync Settings page.
2. From the list of objects, search for the object you want to enable for PushTopic sync.
3. Select the drop-down at the end of the row (▾) and click **Enable Push Based Sync**. The Enable/Disable Push Based Sync dialog is displayed.

4. Click the **Push Based Sync** toggle to enable or disable the object for Push Based Sync.
5. Click **Submit**. The list of objects is updated and indicates the object's status under the Push Based Sync column. If enabled, the next time a PushTopic Event occurs, sync profile data is synced for Push Based Sync enabled objects.

Due to limitations imposed by Salesforce you cannot enable more than 50 objects for Push Based Sync. Please be aware of the following:

- This is applicable across all consumer profiles. Meaning that any objects marked for Push Based Sync contribute to the overall limit.
- For objects marked for Push Based Sync in multiple consumer profiles, the object only counts once towards the maximum.
- For complex objects any child or reference objects also marked for Push Based Sync will be counted towards the maximum.

In addition, if you exceed the Salesforce limit for PushTopic events in a 24-hour period (this is determined by your Salesforce edition – see link below) any objects marked for Push Based Sync will sync only as scheduled in the Data Sync Admin. After the 24-hour period expires (measured 24 hours from the initial PushTopic Event) Push Based Sync will resume until the limit is reached again.

For more information on Salesforce PushTopic Events, refer to the following documentation:

- Salesforce PushTopic overview: https://developer.salesforce.com/docs/atlas.en-us.api_streaming.meta/api_streaming/pushtopic_events_intro.htm
- PushTopic Event Allocations: https://developer.salesforce.com/docs/atlas.en-us.api_streaming.meta/api_streaming/limits.htm

Creating Sync Indexes

For objects with significantly large data sets, it may be good to consider creating a sync index so that pricing and configuration data for that object is more easily searchable after it has been synced to one or more consumer endpoints. You can set up indexes for any objects and specify fields that should be included in the index. You can configure indexes when you add an object for sync or when you manage any objects that are already part of the sync profile.

Sync indexes are most applicable for object pricing data subject to callbacks. For more information on configuring callbacks, refer to TurboEngines administrator documentation.

To add an index to a sync object

1. Go to the Sync Settings page.
2. From list of sync objects, search for the object and select the row drop-down (▾) and click **Manage Index**. The Manage Index page is displayed.
3. Click **Add Index**.
4. Enter a unique **Index Name** in the field provided.
5. Click **Next**.
6. Click the chevron (>) to the left of the object name to view the list of fields.
7. Click the check box next to a field name to add it to the index. Click the check box to the left of the Field Name column to select all visible fields. Note that child objects and reference objects can also be selected.
8. Click **Save**. The index is created and the index list is updated to display the new index. The Index Field column displays the added fields in sequence.
9. Click **Back** to return to the Manage Objects page.

Keep the following in mind when creating an index:

- After adding an index, you cannot make any modifications to it. If you need to make changes, delete and recreate the index.
- You can add a maximum of 32 fields (columns) to an index.
- You can create a maximum of 64 indexes per object.
- You cannot create an index for an object that is marked to "Include all fields including future fields" from the UI. See Data Sync Limitations for more information.

To delete a sync index

1. Go to the Sync Settings page.
2. From list of sync objects, search for the object and select the row drop-down (▾) and click **Manage Index**. The Manage Index page is displayed.
3. From the list of indexes, search for the index and select the row drop-down (▾) and click **Delete**. A confirmation dialog is displayed.
4. Click **Yes** to confirm the deletion or **No** to cancel.
5. Click **Submit** to save the updated index to Sync Settings.

Running Data Sync On-Demand

In cases where you need to initiate data sync manually, you perform an on-demand data sync from the Sync Settings page.

Prerequisites:

- Data sync must be enabled for the given consumer profile.
- The consumer profile must include one or more simple or complex objects.

To run data sync manually

1. Go to the **Sync Settings** page.
2. Click **Sync Now**. The data sync process is begun and you are redirected to the Run History tab.
3. Review Run History to view sync details and any associated errors.

> - If a sync is progress when you invoke on-demand sync, the sync is queued and executes after the in-progress sync is complete.
> - If the sync fails for one or more objects, you can retry the sync. Refer to Retrying Data Sync for steps.

Managing Data Sync Frequency

For a given consumer profile, you can set and schedule master data to be synced at specified intervals. You can manage the frequency of data sync operations and set a start date and time for the first scheduled sync to begin.

To set the initial data sync frequency

1. Go to the **Sync Settings** page.
2. Click **Set New Frequency**. The Set Sync Frequency dialog is displayed.
3. From the **New Sync Frequency** drop-down, select the sync frequency.
4. To specify a start date and time for the first sync, click the checkbox next to **Schedule start date and time**.
   a. Select a date from the **Date** field.
   b. Select a time from the **Time** field.
5. Click **Submit**. The frequency for syncing the data for this consumer profile is set. If no date or start time was configured, the first sync is run. The Sync Settings page displays the newly configured frequency, any run currently in progress, and also the date and time of the next scheduled sync.

To change data sync frequency

1. Go to the **Sync Settings** page.
2. Click **Manage Frequency**. The Set Sync Frequency dialog is displayed.
3. From the **New Sync Frequency** drop-down, select the sync frequency.
4. Click **Submit**.

> ⓘ You cannot schedule a specific date and time for the next sync when you are updating a previously set frequency.

Managing Email Notifications

You can enable or disable email notifications from the Data Sync Admin UI. When enabled, an email notification is sent to the applicable recipients a sync fails or is partially successful ('Failed' or 'Partial Success').

> ⓘ **The From email corresponds to the email address specified in "My Email Settings" for the Salesforce profile associated with the Data Sync admin user.**

To configure email notifications

1. Go to the Sync Settings page.
2. From the upper-right hand corner of the Sync Settings tab, select the **settings drop-down** (⚙▾) and click **Email Notification Settings**. The Email Notification Settings dialog is displayed.
3. Click the **Email Notification** toggle to enable or disable email notifications.
4. Enter a valid email address in the **Email** field and click the **Add** (+)icon. The email address is displayed below the field if added successfully. Click the delete icon (✕) to the right of an email address to remove it from the list.
5. Click **Submit**.

> ⓘ You must add at least one valid email address to enable email notifications.

Working with Data Sync Run History

After a manual or scheduled data sync is run, information about the sync is displayed in the **Run History** tab, regardless of whether or not the data sync was successful. You can view and interact with data sync run history to monitor the progress or status of the sync and any errors generated if the sync was unsuccessful.

The Run History table displays the following information.

Column	Description
Sync Id	The Id of a sync operation. Click the link to view object-level details.
Start Date and Time	The date and time the sync operation initiated. You can use the start date to get an understanding of how recently data was pulled from Salesforce to the consumer service.
End Date and Time	The date and time the sync operation completed or failed.
Duration	The duration of the data sync operation.

Column	Description
Status	

Status	Description
In Progress	The data sync operation is currently in progress.
Success	The data sync operation was successful without any errors.
Partial Success	The data sync operation was successful for some objects but failed for one or more objects.
	The data sync operation was successful for some objects but sync failed and validation warning for one or more objects.
	The data sync operation was failed and the validation warning for one or more objects.
Success with Warning	The data sync operation was successful but validation warning for one or more objects.
Failed	The data sync operation failed with one or more errors.
Pushed	The data sync operation was pushed for one or more objects.

You can click on the error (⚠) icon to view a list of errors and suggested actions.

Column	Description
	Whenever a sync fails or is partially successful, an email notification is automatically generated and sent to the appropriate administrator. Email notifications are configurable and can be enabled or disabled. For more information, see Managing Email Notifications.
Type	Indicates whether the sync was **Scheduled**, **Manual**, or **Resync**. A sync is labeled as a Resync in Run History when one of the following actions has triggered a sync: • An admin adds a new field for sync from the UI and selects "Yes" from the checkbox (to sync the entire object and any newly added fields). • A change in metadata is made to a formula field on the object or dependency object and the field is currently marked for sync. • An Admin User icon (👤) is displayed next to the **Type** to identify the admin user who triggered the corresponding data sync. You can click on the icon to display a pop-up with the admin user's username and email.

Only the most recent syncs are displayed on the page on your screen. To view older syncs, scroll down and the list will refresh. You can also use the search function at the top to find a specific sync Id.

To view object level details for a sync operation

1. Go to the **Run History** page.
2. Locate and click the link to the corresponding **Sync Id**. The Details page is displayed.

The Run History Details page displayed the following information.

Column	Description
Object	The object of the record or records that were synced.
Status	The status of the synced object: • Preparing: The object has data has been pulled from Salesforce but is not yet synced. • Successful: One more records were successfully synced. • No Change: No changes were made to records for that object. • Partial Success: The sync was successful but some records failed to sync. • Failed: The sync failed for the object – check for error message(s). • Aborted: An error occurred fetching object data from Salesforce – the operation to sync this object was not run. • Skipped: Sync was skipped for this object. This status is only present for certain objects in initial sync.
Records Sync Status	Displays the number of records processed for that object out of the total number of object records (for example, "50/50" Processed).

Viewing and Evaluating Error Messages

The Run History tab allows you to view sync details for any sync and to investigate errors that have occurred for any during data sync operations in the list. From the list under Run History, you can view basic sync information and error details and suggested action to take. You can also drill down to the object level to view status for any object in the sync and download a list of error messages if needed for troubleshooting purposes.

To view and evaluate error messages in Run History

1. Go to the **Run History** tab.
2. Locate a **Sync Id** with the status **Failed**.
3. Click the **error** (⚠) icon. A dialog is displayed with a list of errors and suggestion actions to take.

The following table summarizes errors that can occur during a sync operation and suggested actions to take to resolve these errors.

Message	Suggested Action (Consumer Admin)
Unable to sync data to the Consumer Database.	Click "Sync Now" on the Sync Settings page to retry data sync. If the problem persists, please contact Conga Technical Support (you will need to provide the Sync Id).
An internal error has occurred with the Data Sync service.	Click "Sync Now" on the Sync Settings page to retry data sync. If the problem persists, please contact Conga Technical Support (you will need to provide the Sync Id).
Unable to fetch Consumer Profile.	Contact Conga Technical Support with the Sync Id.
Unable to create bulk job on Salesforce.	Click "Sync Now" on the Sync Settings page to retry data sync. If the problem persists, please contact Conga Technical Support (you will need to provide the Sync Id).
Unable to get initial results from Salesforce.	Please check that all objects in Sync Settings have read access. If the problem persists, please contact Conga Technical Support (you will need to provide the Sync Id).
Error occurred while retrieving the records from Salesforce.	Please check that all objects in Sync Settings have read access. If the problem persists, please contact Conga Technical Support (you will need to provide the Sync Id).
Unable to sync data to the target endpoint.	Click "Sync Now" on the Sync Settings page to retry data sync. If the problem persists, please contact Conga Technical Support (you will need to provide the Sync Id).
Unable to fetch Data using the target object	Please contact Conga Technical support with the Sync Id.
Unable to sync data to the target endpoint.	Please contact Conga Technical support with the Sync Id.
Unable to delete data from the target endpoint.	Please contact Conga Technical support with the Sync Id.

Message	Suggested Action (Consumer Admin)
SFDC Api Limit exceeded	It may take up to 24 hours to restore SFDC Api limit. After the limit is restored retry the sync. If the problem persists, please contact Conga Technical Support (you will need to provide the Sync Id).
Metadata sync of an object failed	Retry data sync. If the problem persists, please contact Conga Technical Support (you will need to provide the Sync Id).
Consumer profile sync of an object failed	Retry data sync. If the problem persists, please contact Conga Technical Support (you will need to provide the Sync Id).

To download a list of record-level errors

1. Click the **Sync Id** link for a sync with status 'Failed' or 'Partial Success'. The Run Details page is displayed.
2. From the list of objects synced, find the objects with Failed or Partial Success status.
3. Click the check box to include any errors from an object in the error report. Click the check box to the left of the Object column to select all such objects.
4. Click the **Error Messages** button at the top of the Details list to download a collection (*.zip* file) of error messages generated during the sync in JSON format. It generates the .zip file with *Consumer Profile Name_<Sync ID>_<DateTime>.zip* naming convention and It comprises the following files.
    - **Summary File:** Provides a summary of all the errors, reasons, number of records affected for each object, the corresponding number of columns, and resolution steps for each error.
    - **Details Error File:** Provides details at record level in a separate file. This JSON file is created for each object separately.

Retrying Data Sync

When an a sync fails or is partially successful, you have the option to retry the sync operation for each object that had one or more records fail to sync. After each retry, Run History is updated to reflect any additional records that were synced successfully.

To retry a failed sync

1. Go to the Run History page.
2. Find the sync that failed or was partially successful.
3. Click the **Sync Id** to open the object details page.

4. Click the check box next to an object in the list with the status 'Failed' or 'Partial Success'. Repeat for each object you want to retry. You can select all such objects in the list by clicking the check box to the left of the Object column in the header.
5. From the Details page header, click **Retry**. The on-demand sync executes and all selected objects are updated to 'In Progress' status.

   > ⓘ You cannot retry a sync with status "Aborted".

6. After the on-demand sync finishes, the Status field is updated for each object along with the number of records synced. If the sync retry is successful for all objects, the Status field on the Run History page is also updated.

   > ⓘ Note: When you retry a failed sync, no new entry is created on the Run History page. The entry for the original sync is updated to reflect any changes.

Appendices

Refer to the following appendices for additional important information about TurboEngines Data Sync.

- Data Sync Limitations
- Data Sync FAQ

Data Sync Limitations

The following known issues apply to TurboEngines Data Sync up to the current release (Winter 2020). Please contact Conga Technical Support with any additional questions:

Issue: Adding Cross-Object Formula Fields

Data sync only supports up to eight levels in a cross-object formula field. Any levels beyond eight will be ignored.

Issue: Some Out-of-the-box Objects and Fields Can Cause Sync to Fail

Assumptions: Some objects exist in the list of objects to add that do not support Salesforce "PK chunking" for bulk data queries.

Scenario: Some SFDC out-of-the-box objects in the "Add Objects" list under Sync Settings could cause the data sync to fail if they are added to the consumer profile. There is no effect on custom objects (Conga and customer-specific) as they all support "PK chunking."

Workaround: As a workaround Conga can add objects as they are needed from the back end. Please contact Conga Technical Support if you have any questions regarding this limitation.

Issue: Deleting a Consumer Profile Field in SFDC

Assumptions: Changes are made to the data model for a consumer profile object in SFDC.

Scenario: When a field or related field of an object that is part of a consumer profile is removed, data sync will mark that field as inactive. If it is added back again with the same name, the status of the field remains inactive.

Workaround: Contact Conga Technical Support.

Data Sync FAQ

Q: How long does it take for the next scheduled data sync to complete?

A: This depends on the frequency and the volume of data being completed. The delay between syncs will always be based on the scheduled frequency plus the amount of time to sync the data.

Q: When I delete a field, reference, or object in Sync Settings, is the same data deleted from the consumer endpoint?

A: When you delete a field, reference, or object in Sync Settings, it is no longer enabled for sync. This has no effect on the data at the consumer endpoint other than that any new data for that field or object will no longer be synced.

Q: I have already added an object in sync settings but I want to add one or more fields? How can I be sure that the field will be synced the next time the sync is run?

A: Use the "Manage" feature for an object in sync settings to add additional fields or reference and child fields to the object and save. The next time the sync is run, existing data for those fields are synced to the consumer endpoint.

Q: Can I set up email notifications to send an email to the administrator when a sync completes successfully?

A: No. Email notifications are only sent on partial success or failure of a sync.

Q: As an end-user, I have changed the data and I want to quote the same in Turbo immediately?

A: No. Please check your data sync scheduler. You must complete the sync before using the Turbo functionality for quoting.

REST API Guide

Select one of the following topics for more information:

About REST API Guide

This guide provides an API reference for the TurboConfig Publisher API. Use this API to sync product configuration data from Salesforce to the TurboConfig product configuration engine for processing. You can sync product data to TurboConfig by bundle Id or category Id.

Topic	Description
What's Covered	This guide provides a reference for the TurboConfig Apttus Publisher REST API provided by Apttus.
Primary Audience	• Apttus TurboEngines Administrators • API Developers • Customer Administrators
IT Environment	Refer to the latest *Apttus TurboEngines Summer 2020 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 What's New topic.
Other Resources	• *Apttus TurboEngines Summer 2020 Administrator Guide* • Apttus *CPQ on Salesforce Administrator Guide.*

This guide describes the following task:

- Syncing product data using TurboConfig Publisher API.

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

- Understanding of REST architecture
- Knowledge of REST request and response payloads and operations

- Knowledge of Salesforce API calls
- Salesforce and Apttus terms and definitions
- Basic understanding of Apttus TurboConfig

DOC ID: CTESPR21APIG20210303

What's New in API Guide

The following table lists changes in documentation to support each release.

Document	Publication Date	Topic	Description
Summer '21	06 Jul 2021	N/A	No new APIs were introduced in this release. The guide was updated to reflect product name changes.
Spring '21		N/A	No new APIs were introduced in this release. The guide was updated to reflect product name changes.
Winter 2020		TurboConfig Publisher	Deleted Topic. A new Data Sync Service is introduced in this release. It provides a high-performance mechanism to sync master data at regular, scheduled intervals (or on-demand) between Conga CPQ on Salesforce and specific consumer endpoints using Conga's **Flexible Computing Platform**.
Summer 2020		All topics.	First release.

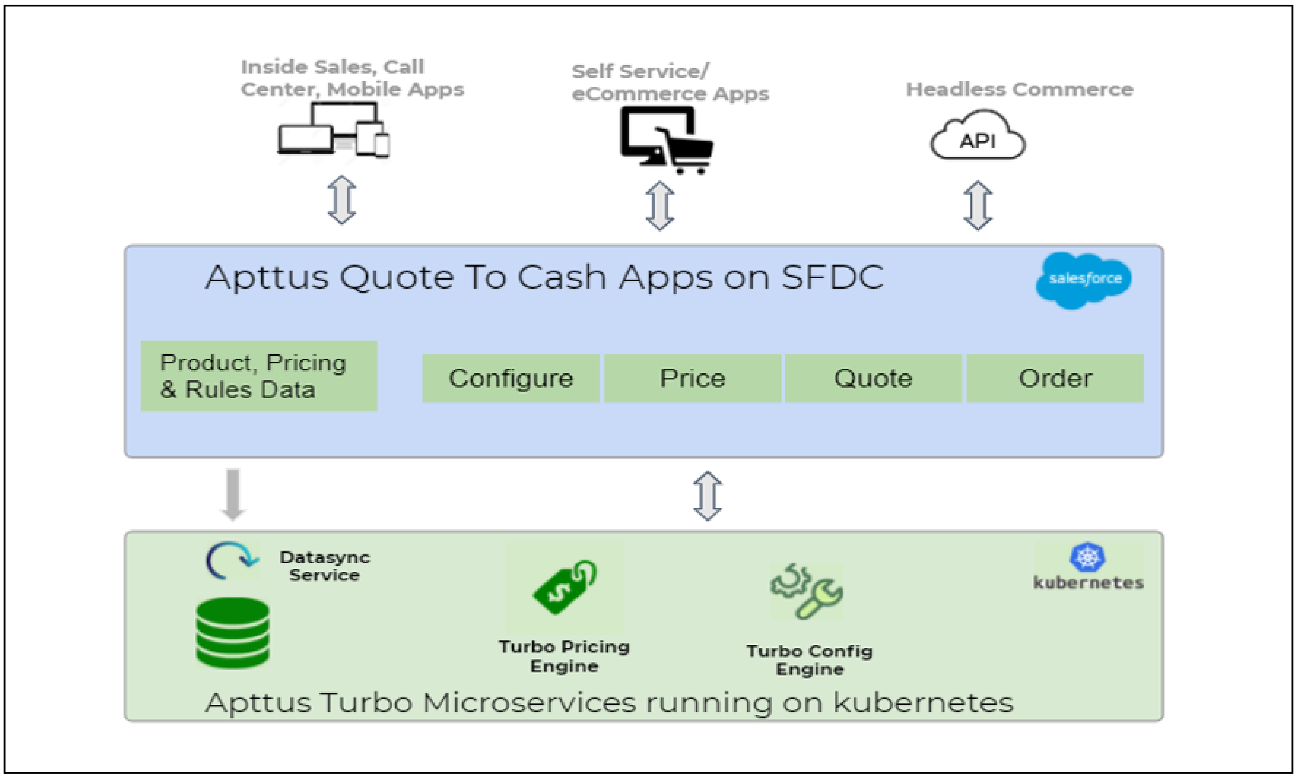
About TurboEngines - Rest API

Conga **TurboEngines** is a concurrent processing engine provided by Conga that comprises various microservices that process product configurations (**TurboConfig**), pricing calculations (**TurboPricing**), and other product-related business data, such as promotions.

Conga TurboEngines offloads the computation workload from the Salesforce platform to the **Conga Flexible Compute Platform** to reduce the processing time on the cart. Processing the computation workload in the Conga Flexible Compute Platform reduces the interaction costs and the quote turnaround time specifically during peak load or large transactions.

TurboEngines scale on the following dimensions:

- Number of users
- Size of transaction
- The complexity of the product and rules


Inside Sales, Call Center, Mobile Apps
Self Service/ eCommerce Apps
Headless Commerce
API
Apttus Quote To Cash Apps on SFDC
salesforce
Product, Pricing & Rules Data
Configure
Price
Quote
Order
Datasync Service
Turbo Pricing Engine
Turbo Config Engine
kubernetes
Apttus Turbo Microservices running on kubernetes

Turbo DataSync APIs

(Open API documentation is only available to view online)

TurboEngines Feature by Release

Review the latest TurboEngines Features by Release document.

- Features by Release

Features by Release

This document contains an overview of features introduced in each major release of Conga TurboEngines. For more information, see TurboEngines Features by Release.

Community & Learning Center Resources

The Conga Customer Community is your one-stop shop for success!

After registering as a new member you'll gain access to a wide variety of resources, including: further details regarding the onboarding process, access to the knowledge base, and product-specific discussion groups.

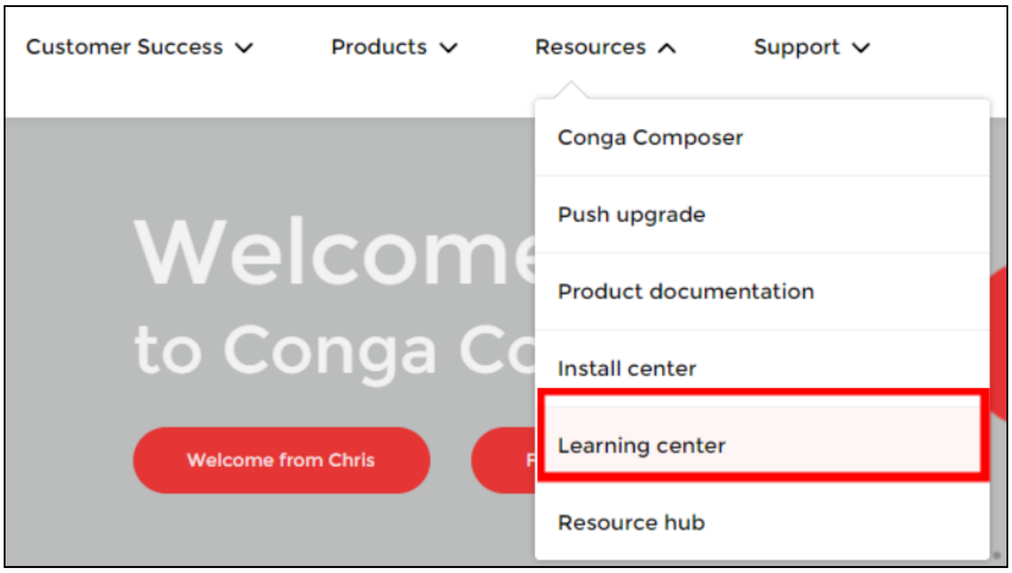
In addition, just a click away from Community is the Conga Learning Center: our comprehensive training portal well stocked with FREE self-paced courses complimented by virtual instructor-led sessions for more advanced and intensive training!

Ready to get started?

First download the Salesforce Authenticator app; you'll need it as part of the login procedure.

Then create a Community account, or log in here!

Once you're logged in, navigate to the Resources dropdown menu and click Learning Center to reach the Conga Learning Center!


Customer Success
Products
Resources
Support
Conga Composer
Push upgrade
Product documentation
Install center
Learning center
Resource hub
Welcome to Conga Co
Welcome from Chris