Consumer Goods Cloud Real Time Reporting Developer Guide

Version 64.0, Summer '25

Last updated: July 25, 2025

CONTENTS

CHAPTER 1 Introduction

This document explains the Salesforce configurations required to export CSV files from within Real Time Reporting (RTR) using the Integration API, and provides information on customizing the RTR using APEX. RTR exports support the export of BOM components.

EDITIONS

Available in: Lightning Experience in **Enterprise** and **Unlimited** Editions that have Consumer Goods Cloud Trade Promotion Management enabled.

CHAPTER 2 Exporting the KPIs from Hyperforce

In this chapter ...

This section provides information on how to export the KPIs from Hyperforce. You can create CSV export integration metadata configuration by adding new report configuration entry to RTR Report Configurations.

Prerequisites

- Consumer Goods Cloud Retail Execution and Trade Promotion Management.
- Salesforce Connected App (for user authentication). See Connected Apps in *Salesforce Help*.
- Integration API user with appropriate rights (that is, read files) configured.
- Ensure to activate Business Years for all the export functions. Export functions are based on the sales org, business year, and data source (e.g. account, promotion).

EDITIONS

Available in: Lightning Experience in **Enterprise** and **Unlimited** Editions that have Consumer Goods Cloud Trade Promotion Management enabled.

Configuring Integration Dimension Meta

Create RTR report configuration records to export CSVs.

There's a object called RTR Report Configuration—`CGCloud_RTR_Report_Configuration__c`—available in your Salesforce org. You must create RTR report configuration records to export CSVs as they hold the dimension meta information for the integration export.

Note: The user mapped to Calculation Result Export Processing Service should have access to the fields and objects configured in report dimensions. The TPM Calculation Result Export permission set provides access to product, account, promotion, tactic, and product part objects and fields. An administrator can assign the TPM Calculation Result Export permission set for this user and also provide access to the additional objects that are used in report dimensions.

EDITIONS

Available in: Lightning Experience in **Enterprise** and **Unlimited** Editions that have Consumer Goods Cloud Trade Promotion Management enabled.

For each dimension and sales org, you must create dimension meta as *`RTR Report Configuration`* records:

- Account Dimension
- Promotion Dimension
- Tactic Dimension
- Product Dimension

Note: When the dimension meta is created and configured, you must save and synchronize the dimension meta to Consumer Goods Cloud Processing Service.

Account Dimension

The account dimension for integration export must be configured for each sales org on which CSV exports is executed.

Promotion Dimension

You must configure the promotion dimension for integration export for each sales org on which CSV exports is executed.

Tactic Dimension

You must configure the tactic dimension for integration exports for each sales org on which CSV exports is executed.

Product Dimension

You must configure product dimension for integration export for each sales org on which CSV exports is executed.

Product Part Dimension

To extract data at the Bill of Materials (BOM) component or product part level, configure Product Part Dimension for each sales org on which CSV exports are executed.

Configuration of Integration Meta Data

The integration meta configuration defines dimensions, export columns, filters, or conditions for CSV exports.

Account Dimension

The account dimension for integration export must be configured for each sales org on which CSV exports is executed.

The integration export fetches content of the account dimension from the Salesforce object—Account. After creation of *`RTR Report Configuration`*" records, provide the following information and a meta JSON:

Attribute	Description
Information Internal Name	A unique internal name of the account.
Configuration Usage	Integration Account Dimension
Configuration of sales org	The sales organization to which the account belongs.

Account dimension meta JSON example

```
[
   {
      "name":"name",
      "fieldsf":"Name",
      "fieldsf$label":"Name"
   },
   {
      "name":"externalid",
      "fieldsf":"CGCloud__ExternalId__c",
      "fieldsf$label":"CGCloud__ExternalId__c"
   },
   {
      "name":"accountnumber",
      "fieldsf":"CGCloud__Account_Number__c",
      "fieldsf$label":"CGCloud__Account_Number__c"
   }
]
```

Promotion Dimension

You must configure the promotion dimension for integration export for each sales org on which CSV exports is executed.

The integration export fetches content of the promotion dimension from the Salesforce object—CGCloud__Promotion__c. After creation of *`RTR Report Configuration`* records, provide the following information and a meta JSON:

Attribute	Description
Information Internal Name	A unique internal name of the promotion.

Attribute	Description
Configuration Usage	Integration Promotion Dimension
Configuration of sales org	The sales organization to which the promotion belongs.

Note: You must include the *accountid* attribute in the dimension referencing on the field—CGCloud__Anchor_Account__c.

Promotion dimension meta JSON example

```
[
   {
      "fieldsf":"Id",
      "name":"id"
   },
   {
      "fieldsf":"CGCloud__Date_From__c",
      "name":"datefrom"
   },
   {
      "fieldsf":"CGCloud__PromotionTemplate__c",
     "fieldsf$label":"CGCloud__Promotion_Template__r.CGCloud__Description_Language_1__c",

      "name":"templateName"
   },
   {
      "fieldsf":"CGCloud__Relevant_For_Account_Plan__c",
      "name":"accountplanrelevant"
   },
   {
      "fieldsf":"CGCloud__Anchor_Account__c",
      "name":"accountid"
   },
   {
      "fieldsf":"CGCloud__Anchor_Account__r.CGCloud__ExternalId__c",
      "name":"accountexternalid"
   }
]
```

Tactic Dimension

You must configure the tactic dimension for integration exports for each sales org on which CSV exports is executed.

The integration export fetches content of the tactic dimension from the Salesforce object—CGCloud__Tactic__c. After creation of *RTR Report Configuration"* records, provide the following information and a meta JSON:

Attribute	Description
Information Internal Name	A unique internal name of the tactic.
Configuration Usage	Integration Tactic Dimension

Attribute	Description
Configuration of sales org	The sales organization to which the tactic belongs.

Tactic dimension meta JSON example

```
[
   {
      "name":"name",
      "fieldsf":"Name",
      "fieldsf$label":"Name"
   },
   {
      "name":"compensationmodel",
      "fieldsf":"CGCloud__Compensation_Model__c",
      "fieldsf$label":"CGCloud__Compensation_Model__c"
   },
   {
      "name":"amount",
      "fieldsf":"CGCloud__Amount__c",
      "fieldsf$label":"CGCloud__Amount__c"
   },
   {
      "name":"lifttype",
      "fieldsf":"CGCloud__Lift_Type__c",
      "fieldsf$label":"CGCloud__Lift_Type__c"
   },
   {
      "name":"liftvalue",
      "fieldsf":"CGCloud__Lift_Value__c",
      "fieldsf$label":"CGCloud__Lift_Value__c"
   }
]
```

Product Dimension

You must configure product dimension for integration export for each sales org on which CSV exports is executed.

The integration export fetches content of the product dimension from the Salesforce object—Product2. After creation of *`RTR Report Configuration"`* records, provide the following information and a meta JSON:

Attribute	Description
Information Internal Name	A unique internal name of the product.
Configuration Usage	Integration Product Dimension
Configuration of sales org	The sales organization to which the product belongs.

Product dimension meta JSON example

```
[
   {
      "fieldsf$label":"CGCloud__Description_1__c",
      "fieldsf":"CGCloud__Description_1__c",
      "name":"description"
   },
   {
      "fieldsf$label":"CGCloud__Consumer_Goods_External_Product_Id__c",
      "fieldsf":"CGCloud__Consumer_Goods_External_Product_Id__c",
      "name":"externalid"
   },
   {
      "fieldsf$label":"CGCloud__Product_Short_Code__c",
      "fieldsf":"CGCloud__Consumer_Goods_Product_Code__c",
      "name":"productcode"
   },
   {
      "fieldsf$label":"CGCloud__Criterion_1_Product_Description__c",
      "fieldsf":"CGCloud__Criterion_1_Product_Description__c",
      "name":"category"
   }
]
```

Product Part Dimension

To extract data at the Bill of Materials (BOM) component or product part level, configure Product Part Dimension for each sales org on which CSV exports are executed.

Note: To save the KPIs related to BOM components in the database, enable the BOM scope in the KPI definition and the BOM Component Writeback option in the storage settings. For more information, see https://help.salesforce.com/s/articleView?id=sf.tpm_foundation_create_kpi_definitions.htm.

The integration export fetches content of the product part dimension from the Salesforce object—`CGCloud__Child_Product__c`. After creation of *`RTR Report Configuration`* records, provide the following information and a meta JSON:

Report Configuration Attribute	Description
Information Internal Name	A unique internal name of the product.
Configuration Usage	Integration Product Part Dimension
Configuration of sales org	The sales organization to which the product belongs.

Integration Product Part Dimension meta JSON example

```
[
   {
      "fieldsf":"Id",
      "name":"id"
   },
   {
```

```
      "name":"quantity",
      "fieldsf":"CGCloud__Quantity__c",
      "fieldsf$label":"CGCloud__Quantity__c"
   },
   {
      "name":"externalid",
      "fieldsf":"CGCloud__ExternalId__c",
      "fieldsf$label":"CGCloud__ExternalId__c"
   },
   {
      "fieldsf":"CGCloud__Child_Product__r.Name",
      "fieldsf$label":"CGCloud__Child_Product__r.Name",
      "name":"componentName"
   }
]
```

Configuration of Integration Meta Data

The integration meta configuration defines dimensions, export columns, filters, or conditions for CSV exports.

Before creating an integration meta configuration entry in *RTR Report Configuration"*, the Salesforce integration dimension meta configuration must be created, saved, and synchronized to off-platform.

Note: If additional custom period is enabled for a salesorg, then the property `enableForCustomTimeLevel: true` should be added to the export configuration.

You can create CSV export integration metadata configuration by adding new report configuration entry to *RTR Report Configurations*.

Note: **It is mandatory to include the key fields of each dimension in the exported columns when exporting a CSV file of the report.**

For CSV export reports, provide the following inputs:

Attribute	Description
Information Internal Name	A unique internal name of the integration metadata configuration.
Configuration Usage	Integration Metadata.
Configuration of sales org	The sales organization to which the product belongs.
Configuration Reporting KPI Set	The KPI set connected to measure dimension.

Note: When you trigger an inbound integration data source, ensure that you specify the *calendaryear* instead of *businessyear* in the request parameters. For more information, see Triggering CSV Exports on page 15.

Examples of the supported data sources

Examples of different supported data sources are listed in this section. Such a meta JSON must be saved to off-platform in order to enable CSV export. A meta JSON consists of the following blocks:

- datasources:

  Inbound datasources:

- PromotionTacticDailyMeasureReal
- PromotionTacticWeeklyMeasureReal
- DailyRealData
- DailyIntData
- ProductMeasures
- AccountMeasures
- AccountProductMeasures

Writeback datasources:

- AccountAndPromotionMeasures
- PromotionMeasures
- AccountAndTacticMeasures

Note: See Triggering the CSV Export Process for more information.

- dimensions:

  Specifies dimensions to fetch data based on the selected data source:

  - accountdimension: Retrieves Salesforce data from CGCloud__Account_Extension__c
  - promotiondimension: Retrieves Salesforce data from CGCloud__Promotion__c
  - tacticdimension: Retrieves Salesforce data from CGCloud__Tactic__c
  - productdimension: Retrieves Salesforce data from Product2
  - kpidimension: Specifies the KPIs that are exported. The KPIs that are calculated based on the KPI set defined in the report configuration are available for the export.
  - timedimension: Specifies how to define the time dimension, by default, it's fixed to the business year.
    - includetotal: true or false (include the total value of the KPI)
    - splitweeks: true (only valid for weekly or custom period) or false
    - period: weekly, monthly, custom, customweek, and custommonth. An additional custom period-enabled salesorg supports only custom, customweek, and custommonth whereas a custom period-enabled salesorg supports custom and monthly period types only. An org using a standard calendar supports only weekly and monthly period types.

      Note: For period type customweek/custommonth, it is mandatory to include the timedimension.StartDate as the first timedimension column.

- exportsettings:

  Specifies columns to generate in CSV file and CSV separator

  - columns: Array of strings referencing dimension attributes.

    dimensionname.attribute must exist in the corresponding dimension. the allowed accessors for timedimension:

    - timedimension.label
    - timedimension.StartDate
    - timedimension.EndDate
    - timedimension.yearnumber

    The allowed accessors for kpidimension:

    - kpidimension.label

- kpidimension.measurecode
- kpidimension.name

Spread operator: The last column requires spread operator on either a kpidimension or timedimension accessor.

Note: When spread operator is used on kpidimension as the last column, accessors of kpidimension can't be used in columns before. When spread operator is used on timedimension as the last column, accessors of timedimension can't be used in columns before.

– Separator: The string specifying CSV separator. The allowed separators are:
  - ,
  - |
  - ;
  - :

- exportfilters:

  Specifies conditions to apply when export process fetches data from Salesforce dimensions.

Triggering the CSV Export Process

The CSV export for a configured integration metadata report is triggered with endpoints provided by the RTR gateway. When the CSV export process is finished, the CSV file can be downloaded through the integration API endpoints.

EDITIONS

Available in: Lightning Experience in **Enterprise** and **Unlimited** Editions that have Consumer Goods Cloud Trade Promotion Management enabled.

Triggering CSV Export
This Apex call expects the mandatory values of the business year, the actual name of the RTR report config record, and the sales org that must be exported.

Checking CSV Export Status
The next step is to check the status of this export.

Committing CSV Export

Retrieving CSV Export
You can fetch the history of the CSV export by making an Apex call.

Getting Access Token for the User

Getting CSV Full Export

Getting CSV Delta Export

Triggering CSV Export

This Apex call expects the mandatory values of the business year, the actual name of the RTR report config record, and the sales org that must be exported.

You can trigger the CSV export by making an Apex call. A maximum of five calls can be triggered simultaneously.

Sample Apex Request

```
<namespace>.OffPlatformCallout request = new
<namespace>.OffPlatformCallout('SCHEDULE_RTR_EXPORT','0001');
Map<String, Object> data = new Map<String, Object>();
data.put('metaname','ReportData0001');
data.put('salesorg', '0001');
data.put('businessyear',2022);

String requestBodyContent = JSON.serialize(data, true);
Map<String, String> params = new Map<String, String>();

<namespace>.OffPlatformCalloutResponse response =
request.execute(<namespace>.TransactionHandler.getTransactionIdentifier(), params,
requestBodyContent);
System.debug(response);
```

Sample Response

The response contains a unique identifier for this export in JSON format. This GUID is needed in the subsequent calls.

```
{
  "csvGuid": "94f49a3f-37bc-4794-8051-123456789"
}
```

Checking CSV Export Status

The next step is to check the status of this export.

As soon as it's ready (that is, "Status" = "Ready"), you can continue with the actual download of the file.

Sample Apex Request:

```
<namespace>.OffPlatformCallout request = new
<namespace>.OffPlatformCallout('GET_RTR_EXPORT_STATUS','0001');
Map<String, String> params = new Map<String, String>();
params.put('csvGuid', '<CSV Guid received as response from Triggering CSV Export>');
<namespace>.OffPlatformCalloutResponse response =
request.execute(<namespace>.TransactionHandler.getTransactionIdentifier(), params, null);
```

Sample Response

```
{
  "Status": "Ready",
  "RequestDate": "2021-09-14T13:33:29.000Z",
  "MetaName": "AccountAndPromotionMeasures",
  "BusinessYear": 2020,
  "Statistics": {
    "csvGuid": "94f49a3f-37bc-4794-8051-123456789",
    "jobStartTime": "Tue, 14 Sep 2021 13:34:02 GMT",
    "hasDelta": false,
    "deltaToCsvGuid": null,
    "metaName": "AccountAndPromotionMeasures",
```

```
    "metaVersion": 8,
    "totalAccounts": 1,
    "totalCategories": 6,
    "dimDataRetrievalDuration": 1188,
    "numRecords": 0,
    "numAccountCategoryKeys": 6,
    "runDuration": 1592,
    "processedAccountCategories": 6,
    "jobCompletionTime": "Tue, 14 Sep 2021 13:34:06 GMT",
    "jobDuration": 4734
  },
  "FullExport": "tpm/long/export/full/94f49a3f-37bc-4794-8051-396f545500eb.csv.gz",
  "DeltaExport": "tpm/long/export/delta/94f49a3f-37bc-4794-8051-396f545500eb.csv.gz"
}
```


Committing CSV Export

Generally, there are two types of CSV exports generated by the system—a full and a delta export.

- The full data contains all the data for the provided business year, report, and sales org.
- The delta export contains only the changed data for the provided business year, report, and sales org since the last commit.

Sample Apex Request

```
<namespace>OffPlatformCallout request = new
<namespace>.OffPlatformCallout('COMMIT_RTR_EXPORT','<salesorg>');
Map<String, Object> data = new Map<String, Object>();
data.put('metaname','ReportData0001');
data.put('salesorg', '0001');
data.put('businessyear',2020);
String requestBodyContent = JSON.serialize(data, true);
Map<String, String> params = new Map<String, String>();
params.put('csvGuid', '<CSV Guid received as response from Triggering CSV Export>');
<namespace>.OffPlatformCalloutResponse response =
request.execute(<namespace>.TransactionHandler.getTransactionIdentifier(), params,
requestBodyContent);
System.debug(response);
```

Sample Response


```
{
  "csvGuid": "94f49a3f-37bc-4794-8051-123456789",
  "commitdate": "Thu, 16 Sep 2021 13:04:23 GMT"
}
```


Retrieving CSV Export

You can fetch the history of the CSV export by making an Apex call.

Sample Apex Request

```
<namespace>.OffPlatformCallout request = new
```

```
<namespace>.OffPlatformCallout('GET_RTR_EXPORT_HISTORY','0001');
Map<String, String> params = new Map<String, String>();
params.put('metaname', 'ReportData0001');
<namespace>.OffPlatformCalloutResponse response =
request.execute(<namespace>.TransactionHandler.getTransactionIdentifier(), params, null);

System.debug(response);
```

Sample Response


```
[
  {
    "csvGuid": "e9b73d84-2a08-4815-a6ff-88a787dc007a",
    "metaname": "ReportData0001",
    "metaversion": 1,
    "businessyear": 2022,
    "salesorg": "0001",
    "status": "Ready",
    "fullExportFile": "full/e9b73d84-2a08-4815-a6ff-88a787dc007a.csv.gz",
    "deltaExportFile": "delta/e9b73d84-2a08-4815-a6ff-88a787dc007a.csv.gz",
    "fileexpiry": "2022-06-15T08:20:35.000Z",
    "commitdate": "2022-05-16T08:24:53.000Z",
    "deltaAgainst": "c32a1252-ee4d-457d-98b2-3f72029c1722",
    "statistics": {
      "csvGuid": "e9b73d84-2a08-4815-a6ff-88a787dc007a",
      "jobStartTime": "Mon, 16 May 2022 08:21:24 GMT",
      "hasDelta": true,
      "deltaToCsvGuid": "c32a1252-ee4d-457d-98b2-3f72029c1722",
      "metaName": "ReportData0001",
      "metaVersion": 1,
      "totalAccounts": 1,
      "totalCategories": 6,
      "dimDataRetrievalDuration": 1401,
      "numRecords": 1392,
      "numAccountCategoryKeys": 6,
      "runDuration": 2032,
      "processedAccountCategories": 6,
      "jobCompletionTime": "Mon, 16 May 2022 08:21:27 GMT",
      "jobDuration": 2441
    }
  }
]
```


Getting Access Token for the User

To perform subsequent requests to the Integration API, you must get an access token for accessing the Integration API. You can get access tokens in multiple ways before sending requests to the RTR API endpoints. This example uses the simple username-password combination (not recommended in non-DEV environments). The response contains "access_token", which is needed for subsequent requests.

cURL example:

```
curl --location --request POST 'https://login.salesforce.com/services/oauth2/token' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--header 'Cookie: BrowserId=S0x8SC_iEeu04d3AnIIK0w; CookieConsentPolicy=0:0' \
--data-urlencode 'grant_type=password' \
--data-urlencode 'client_id={{CONNECTED APP CLIENT ID}}' \
--data-urlencode 'client_secret={{CONNECTED APP CLIENT SECRET}}' \
--data-urlencode 'username={{USERNAME}}' \
--data-urlencode 'password={{PASSWORT}}'
```

Sample Response:

```
{
"access_token": "xyz!xyz",
"instance_url": "xyz",
"id": "https://login.salesforce.com/id/xyz/xyz",
"token_type": "Bearer",
"issued_at": "1631625949723",
"signature": "xyz"
}
```

Getting CSV Full Export

After you've retrieved the access token, the actual full export file can be requested.

cURL example:

```
curl --location --request GET
'https://int-vir-us.consumergoodscloudprocessingservices.com/api/v1/files/getcsv/{{csvGuid}}?path=full
--header 'Content-Type: application/json' \
--header 'Authorization: Bearer {{access_token}}'
```

The response is a binary file that must be stored. The format of this file is `<filename>.csv`. This archive can then be opened with a compression software.

Getting CSV Delta Export

Once the actual export file is requested, the CSV delta export file can be requested.

cURL example:

```
curl --location --request GET
'https://int-vir-us.consumergoodscloudprocessingservices.com/api/v1/files/getcsv/{{csvGuid}}?path=delta
--header 'Content-Type: application/json' \
--header 'Authorization: Bearer {{access_token}}'
```

The response is a binary file that must be stored. Its format is `<filename>.csv`. This archive can then be opened with a compression software.

REST API Endpoints

This section describes the different REST API endpoints.

EDITIONS

Available in: Lightning Experience in **Enterprise** and **Unlimited** Editions that have Consumer Goods Cloud Trade Promotion Management enabled.

Triggering CSV Exports
Trigger five CSV exports simultaneously. The CSV exports are generated asynchronously in the backend. A subsequent endpoint provides the status of the generation.

Checking Job Status
This endpoint responds with the status of the triggered CSV Export, represented by its `csvGuid`.

Committing CSV
This endpoint is used to commit a specific `csvGuid` and take the related CSV as a new baseline for future delta exports—that is, after this commit, the next delta export contains only those KPIs that have been either added, changed, or deleted from this committed state.

Retrieving CSV Exports

Triggering CSV Exports

Trigger five CSV exports simultaneously. The CSV exports are generated asynchronously in the backend. A subsequent endpoint provides the status of the generation.

Endpoint

`SCHEDULE_RTR_EXPORT`

Request Parameters

Parameter	Mandatory	Type
`businessyear`	Yes	Integer
`metaname`	Yes	String
`salesorg`	Yes	String

Note: When you trigger an inbound integration data source, ensure that you specify `calendaryear` instead of `businessyear` in the request parameters.

Sample Request Body

```
<namespace>.OffPlatformCallout request = new
<namespace>.OffPlatformCallout('SCHEDULE_RTR_EXPORT','0001');
Map<String, Object> data = new Map<String, Object>();
data.put('metaname','AccountAndPromotionMeasures');
```

```
data.put('salesorg', '0001');
data.put('businessyear',2022);

String requestBodyContent = JSON.serialize(data, true);
Map<String, String> params = new Map<String, String>();

<namespace>.OffPlatformCalloutResponse response =
request.execute(<namespace>.TransactionHandler.getTransactionIdentifier(), params,
requestBodyContent);
System.debug(response);
```

Sample Response Body

```
{
  "csvGuid": "94f49a3f-37bc-4794-8051-123456789"
}
```

Checking Job Status

This endpoint responds with the status of the triggered CSV Export, represented by its `csvGuid`.

The status can have one of the following values:

- Queued
- InProgress
- Ready
- Error

Endpoint

`GET_RTR_EXPORT_STATUS`

Request Parameters

URL Parameter	Sample Value
`csvGuid`	3b549543-d420-401a-b424-29c7f3c75713

Sample Response Body

```
<namespace>.OffPlatformCallout request = new
<namespace>.OffPlatformCallout('GET_RTR_EXPORT_STATUS','0001');
Map<String, String> params = new Map<String, String>();
params.put('csvGuid', '<CSV Guid received as response from Triggering CSV Export>');
<namespace>.OffPlatformCalloutResponse response =
request.execute(<namespace>.TransactionHandler.getTransactionIdentifier(), params, null);
```

Committing CSV

This endpoint is used to commit a specific `csvGuid` and take the related CSV as a new baseline for future delta exports—that is, after this commit, the next delta export contains only those KPIs that have been either added, changed, or deleted from this committed state.

You can see the current version of this commit when checking the status of the export. For example: `"metaVersion": 8`.

If the underlying RTR Report Configuration record in Salesforce changes, It's reflected as an implicit commit and therefore, set as a new baseline.

Endpoint

`COMMIT_RTR_EXPORT`

Request Parameters

URL Parameter	Sample Value
`csvGuid`	3b549543-d420-401a-b424-29c7f3c75713

Parameter	Mandatory	Type
`businessyear`	Yes	Integer
`metaname`	Yes	String
`salesorg`	Yes	String

Sample Request Body

```
<namespace>OffPlatformCallout request = new
<namespace>.OffPlatformCallout('COMMIT_RTR_EXPORT','<salesorg>');
Map<String, Object> data = new Map<String, Object>();
data.put('metaname','AccountAndPromotionMeasures');
data.put('salesorg', '0001');
data.put('businessyear',2020);
String requestBodyContent = JSON.serialize(data, true);
Map<String, String> params = new Map<String, String>();
params.put('csvGuid', '<CSV Guid received as response from Triggering CSV Export>');
<namespace>.OffPlatformCalloutResponse response =
request.execute(<namespace>.TransactionHandler.getTransactionIdentifier(), params,
requestBodyContent);
System.debug(response);
```

Sample Response Body

```
{
  "csvGuid": "94f49a3f-37bc-4794-8051-123456789",
  "commitdate": "Thu, 16 Sep 2021 13:04:23 GMT"
}
```

Retrieving CSV Exports

The response body is a binary file and can be saved locally. This file with the ending `csv.gz` is a zipped CSV file, so it must be extracted first.

Endpoint

```
<CGCloud Processing Services Integration>/api/v1/files/getcsv/{{csvGuid}}?path={{full or delta}}
```

Note: You can get the CGCloud Processing Services Integration value from the Processing Services Pairing App page in your Salesforce org.

HTTP Method

GET

URL Parameter	Mandatory	Sample Values
`csvGuid`	Yes	3b549543-d420-401a-b424-29c7f3c75713
`path`	Yes	Must be either: • tpm/long/export/full (to retrieve a full export) • tpm/long/export/delta (to retrieve a delta export)

Examples

This section provides sample code snippets for different data sources.

Available in: Lightning Experience in **Enterprise** and **Unlimited** Editions that have Consumer Goods Cloud Trade Promotion Management enabled.

Data Source: Account and Promotion Measures

This section provides sample code snippets for different data sources.

Data Source: Promotion Measures

An integration metadata report configuration using the data source `PromotionMeasures` requires dimensions.

Data Source: Promotion Measures with BOM Components

The `PromotionMeasuresWithBOMComponents` data source allows export of information related to promotion measures at the Bill of Material (BOM) component or product part level in RTR.

Data Source: Promotion and Tactic Measures

An integration metadata report configuration using the data source `PromotionAndTacticMeasures` requires dimensions.

Data Source: Promotion and Tactic Measures with BOM Components

The `PromotionAndTacticMeasuresWithBOMComponents` data source allows export of information related to promotion and tactic measures at the Bill of Material (BOM) component or product part level in RTR.

Data Source: DailyRealData/DailyIntData

This section provides sample code snippets for the integration metadata JSON with the data source. The following example shows the datasources configuration payload for *`DailyRealData/dailyIntData`*.

Data Source: AccountMeasure

This example provides details on the integration metadata JSON with the data source. This example provides details on the integration metadata JSON with the data source. The following example shows the datasources configuration payload for *`AccountMeasure`*.

Data Source: PromotionTacticDailyMeasureReal

This example provides details on the integration metadata JSON with the data source. This example provides details on the integration metadata JSON with the data source. The following example shows the datasources configuration payload for *`PromotionTacticDailyMeasureReal`*.

Data Source: PromotionTacticWeeklyMeasureReal

This example provides details on the integration metadata JSON with the data source. This example provides details on the integration metadata JSON with the data source. The following example shows the datasources configuration payload for *`PromotionTacticWeeklyMeasureReal`*.

Data Source: AccountProductMeasure

This example provides details on the integration metadata JSON with the data source. The following example shows the datasources configuration payload for *`AccountProductMeasure`*.

Data Source: ProductMeasure

This section provides sample code snippets for the integration metadata JSON with the data source. The following example shows the datasources configuration payload for *`ProductMeasure`*.

Data Source: WeeklyMeasureReal

This section provides sample code snippets for the integration metadata JSON with the data source. The following example shows the datasources configuration payload for *`WeeklyMeasureReal`*.

Data Source: WeeklyMeasureInt

This section provides sample code snippets for the integration metadata JSON with the data source. The following example shows the datasources configuration payload for *`WeeklyMeasureInt`*.

Data Source: Account and Promotion Measures

This section provides sample code snippets for different data sources.

An integration metadata report configuration using the data source `AccountandPromotionMeasures` requires the following dimensions:

- `accountdimension`
- `promotiondimension`
- `productdimension`
- `kpidimension`
- `timedimension`

Example: Spreading Time Label

This example provides details on the integration metadata JSON with datasource AccountAndPromotionMeasures (time label spreaded).

```
{
  "datasources": [
    {
      "name": "AccountAndPromotionMeasures"
    }
  ],
  "dimensions": {
    "accountdimension": {
      "attributes": [
        {
          "label": "ExternalId",
          "name": "externalid"
        },
        {
          "label": "Name",
          "name": "name"
        },
        {
          "label": "AccountNUmber",
          "name": "accountnumber"
        }
      ],
      "key": "externalid",
      "level": ""
    },
    "productdimension": {
      "attributes": [
        {
          "label": "Description",
          "name": "description"
        },
        {
          "label": "ExternalId",
          "name": "externalid"
        },
        {
          "label": "Category",
          "name": "category"
        }
      ],
      "key": "externalid",
      "level": "Product"
    },
    "timedimension": {
      "includetotal": false,
      "splitweeks": false,
```

```
    "periodtype": "custom"
  },
  "kpidimension": {
    "measures": [
      {
        "name": "ProPlanIncrVolume",
        "label": "Incremental Volume"
      },
      {
        "name": "PlanTotalVolumeResult",
        "label": "Planned Baseline"
      }
    ]
  },
  "promotiondimension": {
    "attributes": [
      {
        "label": "id",
        "name": "id"
      },
      {
        "label": "accountid",
        "name": "accountid"
      },
      {
        "label": "accountplanrelevant",
        "name": "accountplanrelevant"
      }
    ],
    "key": "id",
    "level": ""
  }
},
"exportsettings": {
  "columns": [
    "accountdimension.externalid",
    "productdimension.externalid",
    "productdimension.category",
    "productdimension.description",
    "kpidimension.label",
    "kpidimension.measurecode",
    "...timedimension.label"
  ],
  "csvseparator": ","
},
"exportfilters": {
  "conditions": [
    {
      "operator": "includes",
      "value": [
        "Kroger_Atlanta"
      ],
      "attribute": "externalid",
      "dimension": "accountdimension",
```

```
        "name": "cond1"
      }
    ]
  }
}
```

Data Source: Promotion Measures

An integration metadata report configuration using the data source `PromotionMeasures` requires dimensions.

Here is a list of dimensions:

- `accountdimension`
- `promotiondimension`
- `productdimension`
- `kpidimension`
- `timedimension`

Example

This example provides details on the integration metadata JSON with data source.

PromotionMeasures

```
{
  "datasources": [
    {
      "name": "PromotionMeasures"
    }
  ],
  "dimensions": {
    "accountdimension": {
      "attributes": [
        {
          "label": "ExternalId",
          "name": "externalid"
        },
        {
          "label": "Name",
          "name": "name"
        },
        {
          "label": "AccountNUmber",
          "name": "accountnumber"
        }
      ],
      "key": "externalid",
```

```
    "level": ""
  },
  "productdimension": {
    "attributes": [
      {
        "label": "Description",
        "name": "description"
      },
      {
        "label": "ExternalId",
        "name": "externalid"
      },
      {
        "label": "Category",
        "name": "category"
      }
    ],
    "key": "externalid",
    "level": "Product"
  },
  "timedimension": {
    "includetotal": false,
    "splitweeks": false,
    "periodtype": "custom"
  },
  "kpidimension": {
    "measures": [
      {
        "name": "ProPlanIncrVolume",
        "label": "Incremental Volume"
      }
    ]
  },
  "promotiondimension": {
    "attributes": [
      {
        "label": "id",
        "name": "id"
      },
      {
        "label": "datefrom",
        "name": "datefrom"
      },
      {
        "label": "templateName",
        "name": "templateName"
      },
      {
        "label": "accountid",
        "name": "accountid"
      },
      {
        "label": "accountplanrelevant",
        "name": "accountplanrelevant"
```

```
        },
        {
          "label": "accountexternalid",
          "name": "accountexternalid"
        }
      ],
      "key": "id",
      "level": ""
    }
  },
  "exportsettings": {
    "columns": [
      "accountdimension.externalid",
      "promotiondimension.id",
      "promotiondimension.datefrom",
      "promotiondimension.templateName",
      "productdimension.externalid",
      "productdimension.category",
      "productdimension.description",
      "kpidimension.label",
      "kpidimension.measurecode",
      "...timedimension.label"
    ],
    "csvseparator": ","
  },
  "exportfilters": {
    "conditions": [
      {
        "operator": "includes",
        "value": [
          "Kroger_Atlanta"
        ],
        "attribute": "externalid",
        "dimension": "accountdimension",
        "name": "cond1"
      }
    ]
  }
}
```


Export CSV File


```
accountdimension.externalid,promotiondimension.id,promotiondimension.datefrom,promotiondimension.tem
plateName,
productdimension.externalid,productdimension.category,productdimension.description,label,measurecode
,Period-1

/2021,Period-2/2021,Period-3/2021,Period-4/2021,Period-5/2021,Period-6/2021,Period-7/2021,Period-
8/2021,Period-9

/2021,Period-10/2021,Period-11/2021,Period-12/2021,Period-13/2021,Period-14/2021,Period-
15/2021,Period-16/2021,
```

```
Period-17/2021,Period-18/2021,Period-19/2021,Period-20/2021,Period-21/2021,Period-22/2021,Period-
23/2021,Period-
24/2021,Period-25/2021,Period-26/2021,Period-27/2021,Period-28/2021,Period-29/2021,Period-
30/2021,Period-31

/2021,Period-32/2021,Period-33/2021,Period-34/2021,Period-35/2021,Period-36/2021,Period-
37/2021,Period-38/2021,

Period-39/2021,Period-40/2021,Period-41/2021,Period-42/2021,Period-43/2021,Period-44/2021,Period-
45/2021,Period-
46/2021,Period-47/2021,Period-48/2021,Period-49/2021,Period-50/2021,Period-51/2021,Period-52/2021

Kroger_Atlanta,a2o11000001950rAAA,2021-03-08,Customer
Promotion,Crispy_Caramel_Wafer,Snacks,Upples
Matcha
Chocolate Chip,Incremental Volume,PPIV,,,,,,,,,
1388.888333,1620.369722,1620.369722,231.481389,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
Kroger_Atlanta,a2o11000001950rAAA,2021-03-08,Customer
Promotion,Crispy_Cream_Wafer,Snacks,Upples
Golden Oat
Crumples,Incremental Volume,PPIV,,,,,,,,,
1388.888333,1620.369722,1620.369722,231.481389,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
```

Data Source: Promotion Measures with BOM Components

The `PromotionMeasuresWithBOMComponents` data source allows export of information related to promotion measures at the Bill of Material (BOM) component or product part level in RTR.

Here's a list of dimensions required for the `PromotionMeasuresWithBOMComponents` data source in the integration metadata report configuration:

- `accountdimension`
- `promotiondimension`
- `productdimension`
- `kpidimension`
- `timedimension`
- `productpartdimension`

Example

This sample integration metadata configuration specifies product data at the product part level, and defines the fields from the sObjects to be exported along with the KPIs.

Example

This sample integration metadata configuration specifies product data at the product part level, and defines the fields from the sObjects to be exported along with the KPIs.

PromotionMeasuresWithBOMComponents

```
{
   "datasources":[
```

```
    {
       "name":"PromotionMeasuresWithBOMComponents"
    }
 ],
 "dimensions":{
    "accountdimension":{
       "attributes":[
          {
             "label":"ExternalId",
             "name":"externalid"
          },
          {
             "label":"Name",
             "name":"name"
          },
          {
             "label":"AccountNUmber",
             "name":"accountnumber"
          }
       ],
       "key":"externalid",
       "level":""
    },
    "productdimension":{
       "attributes":[
          {
             "label":"Description",
             "name":"description"
          },
          {
             "label":"ExternalId",
             "name":"externalid"
          },
          {
             "label":"Category",
             "name":"category"
          }
       ],
       "key":"externalid",
       "level":"Product"
    },
    "timedimension":{
       "includetotal":false,
       "splitweeks":false,
       "periodtype":"custom"
    },
    "kpidimension":{
       "measures":[
          {
             "name":"BOMPlanIncrVolAll",
             "label":"Planned Incremental Volume"
          }
       ]
    },
```

```
    "promotiondimension":{
       "attributes":[
          {
             "label":"id",
             "name":"id"
          },
          {
             "label":"datefrom",
             "name":"datefrom"
          },
          {
             "label":"templateName",
             "name":"templateName"
          },
          {
             "label":"accountid",
             "name":"accountid"
          },
          {
             "label":"accountplanrelevant",
             "name":"accountplanrelevant"
          },
          {
             "label":"accountexternalid",
             "name":"accountexternalid"
          }
       ],
       "key":"id",
       "level":""
    }
 },
 "productpartdimension":{
    "attributes":[
       {
          "label":"id",
          "name":"id"
       },
       {
          "label":"quantity",
          "name":"quantity"
       },
       {
          "label":"externalid",
          "name":"externalid"
       }
    ],
    "key":"id",
    "level":""
 },
 "exportsettings":{
    "columns":[
       "accountdimension.externalid",
       "promotiondimension.id",
       "promotiondimension.datefrom",
```

```
            "promotiondimension.templateName",
            "productdimension.externalid",
            "productdimension.category",
            "productdimension.description",
            "productpartdimension.id",
            "productpartdimension.externalid",
            "productpartdimension.quantity",
            "kpidimension.label",
            "kpidimension.measurecode",
            "timedimension.label"
        ],
        "csvseparator":","
    },
    "exportfilters":{
        "conditions":[
            {
                "operator":"includes",
                "value":[
                    "Kroger_Atlanta"
                ],
                "attribute":"externalid",
                "dimension":"accountdimension",
                "name":"cond1"
            }
        ]
    }
}
```

Data Source: Promotion and Tactic Measures

An integration metadata report configuration using the data source `PromotionAndTacticMeasures` requires dimensions.

Here is a list of dimensions:

- `accountdimension`
- `promotiondimension`
- `productdimension`
- `kpidimension`
- `timedimension`

Example
This example provides details on the integration metadata JSON with data source.
Export CSV File

Example

This example provides details on the integration metadata JSON with data source.

PromotionandTacticMeasures


```
{
  "datasources": [
    {
      "name": "PromotionAndTacticMeasures"
    }
  ],
  "dimensions": {
    "accountdimension": {
      "attributes": [
        {
          "label": "ExternalId",
          "name": "externalid"
        },
        {
          "label": "Name",
          "name": "name"
        },
        {
          "label": "AccountNUmber",
          "name": "accountnumber"
        }
      ],
      "key": "externalid",
      "level": ""
    },
    "productdimension": {
      "attributes": [
        {
          "label": "Description",
          "name": "description"
        },
        {
          "label": "ExternalId",
          "name": "externalid"
        },
        {
          "label": "Category",
          "name": "category"
        }
      ],
      "key": "externalid",
      "level": "Product"
    },
    "timedimension": {
      "includetotal": false,
      "splitweeks": false,
      "periodtype": "custom"
    },
    "kpidimension": {
      "measures": [
        {
          "label": "Incremental Volume",
          "name": "ProPlanIncrVolume"
```

```
      },
      {
        "label": "Tactic Spend",
        "name": "PlanTotalPromoCosts"
      },
      {
        "label": "Lump Sum",
        "name": "FixedAmountLumpSum"
      }
    ]
  },
  "promotiondimension": {
    "attributes": [
      {
        "label": "id",
        "name": "id"
      },
      {
        "label": "datefrom",
        "name": "datefrom"
      },
      {
        "label": "templateName",
        "name": "templateName"
      },
      {
        "label": "accountid",
        "name": "accountid"
      },
      {
        "label": "accountplanrelevant",
        "name": "accountplanrelevant"
      },
      {
        "label": "accountexternalid",
        "name": "accountexternalid"
      }
    ],
    "key": "id",
    "level": ""
  },
  "tacticdimension": {
    "attributes": [
      {
        "name": "lifttype",
        "label": "lifttype"
      },
      {
        "name": "name",
        "label": "name"
      },
      {
        "name": "liftvalue",
        "label": "liftvalue"
```

```
        },
        {
          "name": "compensationmodel",
          "label": "compensationmodel"
        }
      ],
      "key": "name",
      "level": ""
    }
  },
  "exportsettings": {
    "columns": [
      "accountdimension.externalid",
      "promotiondimension.id",
      "promotiondimension.datefrom",
      "promotiondimension.templateName",
      "tacticdimension.name",
      "tacticdimension.compensationmodel",
      "productdimension.externalid",
      "productdimension.category",
      "productdimension.description",
      "kpidimension.label",
      "kpidimension.measurecode",
      "...timedimension.label"
    ],
    "csvseparator": ","
  },
  "exportfilters": {
    "conditions": [
      {
        "operator": "includes",
        "value": [
          "Kroger_Atlanta"
        ],
        "attribute": "externalid",
        "dimension": "accountdimension",
        "name": "cond1"
      }
    ]
  }
}
```


Export CSV File


```
accountdimension.externalid,promotiondimension.id,promotiondimension.datefrom,promotiondimension.tem
plateName,
tacticdimension.name,tacticdimension.compensationmodel,productdimension.externalid,productdimension.
category,

productdimension.description,label,measurecode,Period-1/2021,Period-2/2021,Period-3/2021,Period-
4/2021,Period-5

/2021,Period-6/2021,Period-7/2021,Period-8/2021,Period-9/2021,Period-10/2021,Period-11/2021,Period-
```

```
12/2021,

Period-13/2021,Period-14/2021,Period-15/2021,Period-16/2021,Period-17/2021,Period-18/2021,Period-
19/2021,Period-
20/2021,Period-21/2021,Period-22/2021,Period-23/2021,Period-24/2021,Period-25/2021,Period-
26/2021,Period-27

/2021,Period-28/2021,Period-29/2021,Period-30/2021,Period-31/2021,Period-32/2021,Period-
33/2021,Period-34/2021,

Period-35/2021,Period-36/2021,Period-37/2021,Period-38/2021,Period-39/2021,Period-40/2021,Period-
41/2021,Period-
42/2021,Period-43/2021,Period-44/2021,Period-45/2021,Period-46/2021,Period-47/2021,Period-
48/2021,Period-49

/2021,Period-50/2021,Period-51/2021,Period-52/2021
Kroger_Atlanta,a2o11000001950rAAA,2021-03-08,Customer
Promotion,,,Crispy_Caramel_Wafer,Snacks,Upples
Matcha
Chocolate Chip,Incremental Volume,PPIV,,,,,,,,,,
1388.888333,1620.369722,1620.369722,231.481389,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
...
...
...

Kroger_Atlanta,a2o11000001950rAAA,2021-03-08,Customer Promotion,T-
0000064,LumpSum,Crispy_Caramel_Wafer,Snacks,

Upples Matcha Chocolate Chip,Lump Sum,FIXA,,,,,,,,,,
59.52381,69.444444,69.444444,9.920635,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
...
...
...
```

Data Source: Promotion and Tactic Measures with BOM Components

The `PromotionAndTacticMeasuresWithBOMComponents` data source allows export of information related to promotion and tactic measures at the Bill of Material (BOM) component or product part level in RTR.

Here's a list of dimensions required for the `PromotionAndTacticMeasuresWithBOMComponents` data source in the integration metadata report configuration:

- `accountdimension`
- `promotiondimension`
- `productdimension`
- `kpidimension`
- `timedimension`
- `productpartdimension`

Example

This sample integration metadata configuration specifies product data at the product part level.

Example

This sample integration metadata configuration specifies product data at the product part level.

PromotionAndTacticMeasuresWithBOMComponents

```
{
   "datasources":[
      {
         "name":"PromotionAndTacticMeasuresWithBOMComponents"
      }
   ],
   "dimensions":{
      "accountdimension":{
         "attributes":[
            {
               "label":"ExternalId",
               "name":"externalid"
            },
            {
               "label":"Name",
               "name":"name"
            },
            {
               "label":"AccountNUmber",
               "name":"accountnumber"
            }
         ],
         "key":"externalid",
         "level":""
      },
      "productdimension":{
         "attributes":[
            {
               "label":"Description",
               "name":"description"
            },
            {
               "label":"ExternalId",
               "name":"externalid"
            },
            {
               "label":"Category",
               "name":"category"
            }
         ],
         "key":"externalid",
         "level":"Product"
      },
      "timedimension":{
         "includetotal":false,
         "splitweeks":false,
         "periodtype":"custom"
      },
      "productpartdimension":{
```

```
      "attributes":[
         {
            "label":"id",
            "name":"id"
         },
         {
            "label":"quantity",
            "name":"quantity"
         },
         {
            "label":"externalid",
            "name":"externalid"
         }
      ],
      "key":"id",
      "level":""
   },
   "kpidimension":{
      "measures":[
         {
            "label":"Incremental Volume",
            "name":"ProPlanIncrVolume"
         },
         {
            "label":"Tactic Spend",
            "name":"PlanTotalPromoCosts"
         },
         {
            "label":"Lump Sum",
            "name":"FixedAmountLumpSum"
         }
      ]
   },
   "promotiondimension":{
      "attributes":[
         {
            "label":"id",
            "name":"id"
         },
         {
            "label":"datefrom",
            "name":"datefrom"
         },
         {
            "label":"templateName",
            "name":"templateName"
         },
         {
            "label":"accountid",
            "name":"accountid"
         },
         {
            "label":"accountplanrelevant",
            "name":"accountplanrelevant"
```

```
            },
            {
               "label":"accountexternalid",
               "name":"accountexternalid"
            }
         ],
         "key":"id",
         "level":""
      },
      "tacticdimension":{
         "attributes":[
            {
               "name":"lifttype",
               "label":"lifttype"
            },
            {
               "name":"name",
               "label":"name"
            },
            {
               "name":"liftvalue",
               "label":"liftvalue"
            },
            {
               "name":"compensationmodel",
               "label":"compensationmodel"
            }
         ],
         "key":"name",
         "level":""
      }
   },
   "exportsettings":{
      "columns":[
         "accountdimension.externalid",
         "promotiondimension.id",
         "promotiondimension.datefrom",
         "promotiondimension.templateName",
         "tacticdimension.name",
         "tacticdimension.compensationmodel",
         "productdimension.externalid",
         "productdimension.category",
         "productdimension.description",
         "productpartdimension.id",
         "productpartdimension.externalid",
         "kpidimension.label",
         "kpidimension.measurecode",
         "timedimension.label"
      ],
      "csvseparator":","
   },
   "exportfilters":{
      "conditions":[
         {
```

```
            "operator":"includes",
            "value":[
               "Kroger_Atlanta"
            ],
            "attribute":"externalid",
            "dimension":"accountdimension",
            "name":"cond1"
         }
      ]
   }
}
```

Data Source: DailyRealData/DailyIntData

This section provides sample code snippets for the integration metadata JSON with the data source. The following example shows the datasources configuration payload for *`DailyRealData/dailyIntData`*.

```
{

 "datasources": [

 {

 "name": "{{DatasourceName}}"

 }

 ],

 "dimensions": {

 "kpidimension": {

 "measures": [

 {

 "name": "ProPlanIncrVolume",

 "label": "Planned Incr. Volume"

 }

 ]

 }

 },

 "exportsettings": {

 "csvseparator": ",",
```

```
 },

 "exportfilters": {

 "conditions": []
 }
}
```

Data Source: AccountMeasure

This example provides details on the integration metadata JSON with the data source. This example provides details on the integration metadata JSON with the data source. The following example shows the datasources configuration payload for *AccountMeasure*.

```
{

 "datasources": [

 {

 "name": "{{DatasourceName}}"

 }

 ],

 "dimensions": {

 "kpidimension": {

 "measures": [

 {

 "name": "ProPlanIncrVolume",

 "label": "Planned Incr. Volume"

 }

 ]

 }

 },

 "exportsettings": {

 "csvseparator": ",",

 },

 "exportfilters": {
```

```
 "conditions": []
 }
}
```

Data Source: PromotionTacticDailyMeasureReal

This example provides details on the integration metadata JSON with the data source. This example provides details on the integration metadata JSON with the data source. The following example shows the datasources configuration payload for *PromotionTacticDailyMeasureReal*.

```
{

 "datasources": [

 {

 "name": "{{DatasourceName}}"

 }

 ],

 "dimensions": {

 "kpidimension": {

 "measures": [

 {

 "name": "ProPlanIncrVolume",

 "label": "Planned Incr. Volume"

 }

 ]

 }

 },

 "exportsettings": {

 "csvseparator": ",",

 },

 "exportfilters": {

 "conditions": []
 }
}
```

Data Source: PromotionTacticWeeklyMeasureReal

This example provides details on the integration metadata JSON with the data source. This example provides details on the integration metadata JSON with the data source. The following example shows the datasources configuration payload for *`PromotionTacticWeeklyMeasureReal`*.

```
{
 "datasources": [
 {
 "name": "{{DatasourceName}}"
 }
 ],
 "dimensions": {
 "kpidimension": {
 "measures": [
 {
 "name": "ProPlanIncrVolume",
 "label": "Planned Incr. Volume"
 }
 ]
 }
 },
 "exportsettings": {
 "csvseparator": ",",
 },
 "exportfilters": {
 "conditions": []
 }
}
```

Data Source: AccountProductMeasure

This example provides details on the integration metadata JSON with the data source. The following example shows the datasources configuration payload for *AccountProductMeasure*.

```
{
 "datasources": [
 {
 "name": "{{DatasourceName}}"
 }
 ],
 "dimensions": {
 "kpidimension": {
 "measures": [
 {
 "name": "ProPlanIncrVolume",
 "label": "Planned Incr. Volume"
 }
 ]
 }
 },
 "exportsettings": {
 "csvseparator": ",",
 },
 "exportfilters": {
 "conditions": []
 }
}
```

Data Source: ProductMeasure

This section provides sample code snippets for the integration metadata JSON with the data source. The following example shows the datasources configuration payload for *ProductMeasure*.

```
{
 "datasources": [
 {
 "name": "{{DatasourceName}}"
 }
 ],
 "dimensions": {
 "kpidimension": {
 "measures": [
 {
 "name": "ProPlanIncrVolume",
 "label": "Planned Incr. Volume"
 }
 ]
 }
 },
 "exportsettings": {
 "csvseparator": ",",
 },
 "exportfilters": {
 "conditions": []
 }
}
```

Data Source: WeeklyMeasureReal

This section provides sample code snippets for the integration metadata JSON with the data source. The following example shows the datasources configuration payload for *`WeeklyMeasureReal`*.

```
{
 "datasources": [
 {
 "name": "{{DatasourceName}}"
 }
 ],
 "dimensions": {
 "kpidimension": {
 "measures": [
 {
 "name": "ProPlanIncrVolume",
 "label": "Planned Incr. Volume"
 }
 ]
 }
 },
 "exportsettings": {
 "csvseparator": ",",
 },
 "exportfilters": {
 "conditions": []
 }
}
```

Data Source: WeeklyMeasureInt

This section provides sample code snippets for the integration metadata JSON with the data source. The following example shows the datasources configuration payload for *`WeeklyMeasureInt`*.

```
{
 "datasources": [
 {
 "name": "{{DatasourceName}}"
 }
 ],
 "dimensions": {
 "kpidimension": {
 "measures": [
 {
 "name": "ProPlanIncrVolume",
 "label": "Planned Incr. Volume"
 }
 ]
 }
 },
 "exportsettings": {
 "csvseparator": ",",
 },
 "exportfilters": {
 "conditions": []
 }
}
```

CGCloud Namespace

The `CGCloud` namespace provides classes that allow you to customise Real Time Reporting (RTR) data extraction from Apex.

The following are the classes in the `CGCloud` namespace.

EDITIONS

Available in: Lightning Experience in **Enterprise** and **Unlimited** Editions that have Consumer Goods Cloud Trade Promotion Management enabled.

RTRReportResult Class

Use this class to execute the Trade Promotion Management Real Time Reporting (RTR) report and access the report data.

ReportComponent Class

Represents a Real Time Reporting (RTR) report UI component. This is an abstract class and can't be instantiated.

FlatList Class

Represents a Real Time Reporting (RTR) report UI Flatlist component. This class extends the `ReportComponent` class.

FlatListRowIterable Class

Represents a Real Time Reporting (RTR) report UI Flatlist component set of rows. This class implements the `Iterable` interface that allows it to be used in batch processes.

FlatListRow Class

Represents a Real Time Reporting (RTR) report UI Flatlist component single rows.

ScoreCard Class

Represents a Real Time Reporting (RTR) report scorecard component. This class extends the `ReportComponent` class.

RTRReportResult Class

Use this class to execute the Trade Promotion Management Real Time Reporting (RTR) report and access the report data.

Namespace

CGCloud

Example

Here's an example of how to use Apex to run the Trade Promotion Management Real Time Reporting reports.

```
cgcloud.RTRReportResult reportResult = cgcloud.RTRReportResult.execute(
    'Promotion Report',  // Report Name
    '0001',  // Sales Org
    new Map<String, Object> { // Report filters
        'periodmonth' => new Map<String, Object> {
            'start' => 0,
            'total' => 12,
            'year' => 2022
        },
        'accountsfids' => new List<String> {'001SL0000004dfgYAA'},
        'productsfids' => new List<String> {'01tSL00000018oNYAQ'},
```

```
        'promo_templatesfid' => new List<String> {'a2USL0000000js12AA',
'a2USL0000000jrz2AA'},
        'promo_phase' => new List<String> {
            'Planning',
            'Modeling',
            'Committed',
            'ForApproval'
        },
        'productlevel'=> 'product'
    }
);
```

This returns an instance of a `RTRReportResult` object after running the report. Use the methods provided by the `RTRReportResult` class to access the report data.

Filters can be specified for the report execution. The report filters specified match the filters defined in the RTR Report Metadata configuration (RTR Report Configuration SObject record). Depending on the filter type, the expected filter must have a specific structure or type.

singleselect

Here's an example of how the report's filter metadata looks:

```
{
    "type": "singleselect",
    "name": "kpigroup",
    "label": "KPI Set",
    "source": "KPIGroup"
}
```

Here's an example of how the filter using Apex looks:

```
...
    'kpigroup' => 'Plan',
...
```

multiselect

Here's an example of how the report's filter metadata looks:

```
{
    "type": "multiselect",
    "name": "promo_phase",
    "label": "Promotion Phase",
    "source": "PromotionPhase",
    "defaultValue": [
        "Planning"
    ]
}
```

Here's an example of how the filter using Apex looks:

```
...
    'promo_phase' => new List<String> {
        'Planning',
        'Modeling',
        'Committed',
        'ForApproval'
```

```
    },
...
```

periodmonth

Here's an example of how the report's filter metadata looks:

```
{
    "label": "Period",
    "type": "periodmonth",
    "name": "periodmonth"
}
```

Here's an example of how the filter using Apex looks:

Note: The filter must be represented by an object with the `year`, `start`, and `total` properties.

```
...
    'periodmonth' => new Map<String, Object> {
       'start' => 0, // The starting month. A value between 0 (January) and 11 (December)

        'total' => 12, // How many month data to retrieve. A value between 1 and 18
        'year' => 2022 // The starting year
    },
...
```

periodweek

Here's an example of how the report's filter metadata looks:

```
{
    "label": "Period",
    "type": "periodweek",
    "name": "periodweek"
}
```

Here's an example of how the filter using Apex looks:

Note: The filter must be represented by an object with the `year`, `start`, and `total` properties.

```
...
    'periodweek' => new Map<String, Object> {
        'start' => 0, // The starting Week. A value between 0 (First week of the year)
 to 70
        'total' => 12, // How many weeks data to retrieve. A value between 1 and 100
        'year' => 2022 // The starting year
    },
...
```

hidden

Here's an example of how the report's filter metadata looks:

```
{
     "label": "Not Seen",
     "type": "promo_s_textfield",
     "name": "hidden",
     "defaultValue" : "<<FIRST_VALUE>>",
```

```
     "source" : [{
        "label" : "example",
        "value" : "value1"
     }]
}
```

Here's an example of how the filter using Apex looks:

Note: The filter must be represented with a string.

```
...
    'promo_s_textfield' => 'value1',
...
```

fixed

Here's an example of how the report's filter metadata looks:

```
{
     "label": "Fixed Filter",
     "type": "promo_s_textfield2",
     "name": "fixed",
     "defaultValue" : "<<FIRST_VALUE>>",
     "source" : [{
        "label" : "example",
        "value" : "value1"
     }]
}
```

Here's an example of how the filter using Apex looks:

Note: The filter must be represented with a string.

```
...
    'promo_s_textfield2' => 'value1',
...
```

subaccount

Here's an example of how the report's filter metadata looks:

```
{
    "type": "subaccount",
    "name": "subaccountsfids",
    "label": "Sub Accounts",
    "accountfilter": "accountsfids",
    "source": "SubAccounts"
}
```

Here's an example of how the filter using Apex looks:

Note: The filter must be represented with a List of strings.

```
...
    'subaccountsfids' => new List<String> { '001SL0000004dfgYAA' },
...
```

RTRReportResult Methods

The following are methods for `RTRReportResult`.

execute(String name, String salesOrg, Map<String, Object> filters)

Execute a RTR Report and retrieve all the Report data.

Signature

```
static RTRReportResult execute(String name, String salesOrg, Map<String, Object> filters)
```

Parameters

name
Type:String

The name of the RTR report configuration to execute.

salesOrg
Type:String

The Salesforce org name to execute the report on.

filters
Type:String

Report filters as defined in the RTR report metadata.

The filters attribute must align with the expected filters as defined in the RTR reporting configuration.

Return Value

Type:RTRReportResult

getComponent(String name)

Retrieve an instance of the Reporting UI component.

Signature

```
static RTRReportResult execute(String name, String salesOrg, Map<String, Object> filters)
```

Parameters

name
Type:String

The name of the reporting component as defined in the report metadata `uimapping`.

Return Value

Type:ReportComponent

- For the Flatlist component, specify the response to `RTRReportResult.FlatList`
- For the Scorecard component, specify the response to `RTRReportResult.ScoreCard`

ReportComponent Class

Represents a Real Time Reporting (RTR) report UI component. This is an abstract class and can't be instantiated.

Namespace

CGCloud

FlatList Class

Represents a Real Time Reporting (RTR) report UI Flatlist component. This class extends the `ReportComponent` class.

Namespace

CGCloud

Usage

The `RTRReportResult.FlatList` class is used to access Flatlist information from RTR reports.

Example

Here's an example of how to extract the Flatlist data from the `RTRReportResult` object.

```
// Extract the Flatlist from the result
// - The component name must match its 'uimapping' name
// - The result must be casted to the correct type.
cgcloud.RTRReportResult.Flatlist flatlist = (cgcloud.RTRReportResult.Flatlist)
    reportResult.getComponent('FlatList');

// Retrieve the rows Iterable object
// All rows can be retrieved with the parameter-less "getRows"
cgcloud.RTRReportResult.FlatlistRowIterable iterator = flatlist.getRows('Promo-Product');

// Iterate over all the rows
while (iterator.hasNext()) {
    cgcloud.RTRReportResult.FlatlistRow row = iterator.next();
```

```
    // Extract column data. We assume the specified columns are in the report

    // Dimension information can be extracted with <dimension name>.<field>
    String promotionId = String.valueOf(row.getColumnValue('promotiondimension.id'));
    // KPI Value information can be extracted by querying the KPI Name
    Decimal kpiValue = (Decimal) row.getColumnValue('ProPlanIncrVolume');

    system.debug('Values: ' + promotionId + ' ' + kpiValue);
}
```

FlatList Methods

FlatList Methods

The following are methods for `FlatList`.

getRows()

Returns an instance of a `FlatlistRowIterable` object with all the rows contained in the Flatlist.

Signature

```
static RTRReportResult.FlatList getRows()
```

Return Value

Type:FlatlistRowIterable

getRows(String rowTypeFilter)

Returns an instance of a `FlatlistRowIterable` object with all the rows that match the specified row type filter. The available values for the row type filter can be extracted with the `getRowTypes` method.

Signature

```
static RTRReportResult.FlatList getRows(String rowTypeFilter)
```

Parameters

rowTypeFilter
Type:String

The value of the row type to filter the rows.

Return Value

Type:FlatlistRowIterable

getColumns()

Returns a set that includes all the available column names in the Flatlist.

The method returns the following:

- Dimension attributes—Include the name of the dimension and a report field separated by a dot. For example, `productdimension.id`, `promotiondimension.id`, and `promotiondimension.phase`.
- KPI columns—Include the KPI name. For example, ProPlanIncrVolume.

Signature

```
static RTRReportResult.FlatList getColumns()
```

Return Value

Type:Set<String>

getRowTypes()

Returns all the available row type filters present in the Flatlist. The row type filters can be used to filter the dataset.

Signature

```
static RTRReportResult.FlatList getRowTypes()
```

Return Value

Type:Set<String>

FlatListRowIterable Class

Represents a Real Time Reporting (RTR) report UI Flatlist component set of rows. This class implements the `Iterable` interface that allows it to be used in batch processes.

Namespace

CGCloud

FlatlistRowIterable Methods

FlatlistRowIterable Methods

The following are methods for `FlatlistRowIterable`.

hasNext()
Returns true if a record can be retrieved.

next()
Returns the next record in the dataset.

hasNext()

Returns true if a record can be retrieved.

Signature

```
public RTRReportResult.FlatlistRowIterable hasNext()
```

Return Value

Type:Boolean

next()

Returns the next record in the dataset.

Note: If no more records are present in the dataset, this method returns an error.

Signature

```
public RTRReportResult.FlatlistRowIterable hasNext()
```

Return Value

Type:FlatlistRow

FlatListRow Class

Represents a Real Time Reporting (RTR) report UI Flatlist component single rows.

Namespace

CGCloud

FlatListRow Methods

FlatListRow Methods

The following are methods for `FlatListRow`.

getColumnValue(String columnName)
Extracts the column value specified for the column name Row type filters can be used to filter the dataset.

getColumnValue(String columnName)

Extracts the column value specified for the column name Row type filters can be used to filter the dataset.

Signature

```
public RTRReportResult.FlatListRow getColumnValue(String columnName)
```

Parameters

columnName
Type:String

The name of the column whose value is to be retrieved.

The column name has the following format:

- Dimension attributes—Include the name of the dimension and a report field separated by a dot. For example, `productdimension.id`, `promotiondimension.id`, and `promotiondimension.phase`.
- KPI columns—Include the KPI name. For example, ProPlanIncrVolume.

Return Value

Type:Object

ScoreCard Class

Represents a Real Time Reporting (RTR) report scorecard component. This class extends the `ReportComponent` class.

Namespace

CGCloud

Usage

The `RTRReportResult.ScoreCard` class is accessible outside of the managed package.

Example

Here's an example of how to extract the ScoreCard data from the `RTRReportResult` object.

```
// Extract the ScoreCard from the result
// - The component name must match its 'uimapping' name
// - The result must be casted to the correct type.
cgcloud.RTRReportResult.ScoreCard scoreCard = (cgcloud.RTRReportResult.ScoreCard)
    reportResult.getComponent('ScoreCard');

// Extract KPI data. We assume the specified KPI's are in the configured scorecard
Decimal kpiValue = scoreCard.getValue('ProPlanIncrVolume');
```

```
system.debug('Value: ' + kpiValue);
```

ScoreCard Methods

ScoreCard Methods

The following are methods for `ScoreCard`.

getValue(String kpiName)
Returns true if a record can be retrieved.

getValue(String kpiName)

Returns true if a record can be retrieved.

Signature

```
public RTRReportResult.ScoreCard getValue(String kpiName)
```

Parameters

`kpiName`
Type:String

The name of the KPI from which the value is to returned. The key performance indicator (KPI) must be included in the scorecard.

Return Value

Type:Decimal

CHAPTER 3 Customizing Real Time Reporting (RTR) with APEX

In this chapter ...

You can create a fund report with custom APEX data sources to integrate KPI values that are obtained from other sources such as a Salesforce attribute. You can also create reports with custom APEX filters, along with the standard filters.

Although requirements vary across clients and markets, these general requirements apply for different processes:

- **Strategic Planning:** Provides flexible views on business metrics (profit, profit margin, ROI, costs, revenue, and so on) on either a regional or national level (across many or all key accounts) to drive future recommendations for target volumes, target revenues, marketing initiatives, brand initiatives, and pricing.
- **Fund Management:** Provides you with a hollistic view of the budgeting not only within but also across funds prior to, during, and after a financial year, so that you can control the financials on both an overall and granular level.
- **Account Planning:** Provides you with flexible views on a single account to monitor business development within the full P&L (Profit and Loss) for the relevant brands and categories and on a detailed plan product level.
- **Promotion Planning:** Provides overviews on key promotion metrics (profit, profit margin, ROI, costs, revenue, etc.) for a single account and time frame on both summary and detailed levels to conduct pre-event and post-event analysis.
- **Promotion Execution:** Tracks claims against promotions or tactics in promotion-based reports and fund reporting to be always aware of missing claim information and outstanding deductions.
- **Post-Event Analysis:** Evaluates the success of single promotional events, brand and marketing initiatives across accounts, and the overall success rate on account-related contracts and their impact on single accounts.

Real Time Reporting supports these requirements by providing a technical foundation to retrieve the important business metrics and allowing flexible configurations to design reports according to client requirements.

Example: Consider this scenario:

During an iterative planning processes, John—the Key Account Manager (KAM), identifies gaps in promotion planning and makes the relevant changes. Real Time Reporting (RTR) helps him to view the impact on his account directly after releasing the changes, allowing him to create a more fruitful Account Plan for his retail partner. The immediate effect of changes that reflect in the report helps him to act instantly and adapt promotion planning on the fly. With RTR, John can view the impact on a summary level by either brand or category and on a detailed plan product level.

These are the five key functional benefits of RTR:

- Supports the full TPM closed loop with real-time insights to drive the decision-making process.
- Provides a consistent user experience for CG Cloud TPM users.

- Builds the trust of users because the numbers that are shown in RTR and other application areas (for example, P&L sheets) are consistent.
- Supports the export of data in MS-Excel format for all reports that allow users to work further on exported data.
- Allows you to define user-specific views so that users can switch between reports instantly.

Create a Fund Report with Custom Apex Datasources

The data that's required for the reports is available on a Hyperforce server and is interfaced via regular data sources such as AccountMonthlyMeasures or AccountWeeklyMeasures.

If you want to integrate KPI values that's obtained from other sources such as a Salesforce attribute, use Real Time Report (RTR) to add a custom datasource. A custom datasource is an Apex endpoint that delivers KPI values to the report.

RTR offers you these custom data sources:

- RTRSalesforceMonthlyMeasures: Used to integrate monthly values for KPIs.
- RTRSalesforceWeeklyMeasures: Used to integrate weekly values for KPIs.

Depending on your report, you can use either weekly or monthly custom data sources to fetch KPIs for that time level. Ensure that you also implement an Apex endpoint that corresponds to these data sources that can deliver these KPIs in the corresponding time granularity (week or month).

Snippet to Define Custom Datasources

```
"datasources": [
      {
      "name": "AccountMonthlyMeasures"
      },
      {
      "name": "RTRSalesforceMonthlyMeasures"
      }
      ],
```

In addition to defining the datasource in the report configuration, ensure that you also create a base Apex class (based on your use case) and a logic Apex class.

Use Case: Fund Amount

The Fund object stores the fund amount in Salesforce. To show a fund amount for a particular category or brand, fetch it from the Salesforce database by using custom data sources.

Prerequisites

1. Create a KPI definition for the fund amount. For this KPI definition, ensure that the object scope is Account and it has the `writeback` feature enabled.

   Note: This KPI definition only provides a name for the external KPI. The Apex class defines how to input values in this KPI.

2. Enter the name of the datasource table as `Account Measure`.
3. Add the KPI to the Reporting KPI set that's used in the report.

Report Configuration

Apart from defining RTRSalesforceMonthlyMeasures as an additional datasource in the report configuration, ensure that you add the newly created KPI to the list of measures.

```
{
"label": "Fund Amount",
"name": "myFndAmount"
},
```

Apex Classes

The custom data sources are related to two base Apex classes. You can use both classes only for code-routing and not to perform any actual logic, queries, or calculations; this is because all reports share these classes.

Note: In RTR, integration users run both the Apex classes, not the user who opens the report. As a result, integration users must have access permissions to the used Apex classes.

TPM_RTRReportingWrapper_AMS (Base Class)

`TPM_RTRReportingWrapper_AMS` is a base Apex class that receives the payload that's constructed by filter selection in the report. The payload contains the KPIs that are defined in the report, the customer, dates, and other important attributes. It then deserializes the payload into an object that can be used in all other related classes.

TPM_RTRSalesforceMonthlyMeasures_AMS (Base Class)

`TPM_RTRSalesforceMonthlyMeasures_AMS` is one of the base classes that's used for code-routing and is required to retrieve the information by a Salesforce REST endpoint `(/services/apexrest/RTRSalesforceMonthlyMeasures)`.

TPM_RTRFunds_AMS (The Logic Class)

Use RTRSalesforceMonthlyMeasures as the datasource to execute the base classes by every report.

TPM_RTRReportingWrapper_AMS (Base Class)

`TPM_RTRReportingWrapper_AMS` is a base Apex class that receives the payload that's constructed by filter selection in the report. The payload contains the KPIs that are defined in the report, the customer, dates, and other important attributes. It then deserializes the payload into an object that can be used in all other related classes.

Note: Ensure that this wrapper class name is the same as the one in the Fund Amount use case. However, to incorporate this class with the fund amount scenario, modify the class.

Sample Code

```
global with sharing class TPM_RTRReportingWrapper_AMS {
global class Periodmonth {
global Integer year;
global Integer start;
global Integer total;
}
global class InputPayload {
global List<String> accountsfids;
global List<String> productsfids;
global Periodmonth periodmonth;
global String timelevel;
```

```
global List<String> readproductsfids;
global List<String> kpis;
global String responsetype;
global Boolean error;
}
global class OutputRecord {
global String pdim;
global String kdim;
global String tdim;
global Double v;
global OutputRecord(String pdim, String kdim, String tdim, Double v) {
this.pdim = pdim;
this.kdim = kdim;
this.tdim = tdim;
this.v = v;
}
}
}
```

TPM_RTRSalesforceMonthlyMeasures_AMS (Base Class)

`TPM_RTRSalesforceMonthlyMeasures_AMS` is one of the base classes that's used for code-routing and is required to retrieve the information by a Salesforce REST endpoint `(/services/apexrest/RTRSalesforceMonthlyMeasures)`.

The system sends the selected filter criteria and the list of KPIs as input payload to the REST endpoint. The APEX REST method then returns the fund amounts as output records to show in the report.

The fund values are returned by a function that's executed when a POST request is sent.

These steps are run within the Apex function:

1. The payload with filter values is converted into a format that can be used in the Apex code.
2. The time range (start and end date) is built by the selected month information.
3. The function that contains the logic for this use case is invoked.

Sample Code

```
@RestResource(urlMapping='/RTRSalesforceMonthlyMeasures/*')
global with sharing class TPM_RTRSalesforceMonthlyMeasures
{
@httpPost
global static List<TPM_RTRReportingWrapper_AMS.OutputRecord> doPost() {
String requestBody = RestContext.request.requestbody.tostring();
TPM_RTRReportingWrapper_AMS.InputPayload payload = (TPM_RTRReportingWrapper_AMS.InputPayload)
JSON.deserialize (requestBody , TPM_RTRReportingWrapper_AMS.InputPayload.class);
Date inputDateBegin = Date.newInstance(payload.periodmonth.year,payload.periodmonth.start,1);
Date inputDateEnd = inputDateBegin.addMonths( payload.periodmonth.total );
return TPM_RTRFunds_AMS.doPost( inputDateBegin , inputDateEnd, payload);
}
}
```

TPM_RTRFunds_AMS (The Logic Class)

Use RTRSalesforceMonthlyMeasures as the datasource to execute the base classes by every report.

This function performs these steps:

1. Defines a list of output record items.
2. Retrieves the fund amounts from Salesforce data by using SOQL, depending on the use case. In this example, only funds at the category level are shown. You can use the category filter as the selection criterion. The funds are also limited to the account and time frame that's selected in the filter.
3. For every result record, the function creates an output record and adds it to the list of results.

```
global with sharing class TPM_RTRFunds_AMS{
global static List<TPM_RTRReportingWrapper_AMS.OutputRecord>
doPost(Date inputDateBegin , Date inputDateEnd,
TPM_RTRReportingWrapper_AMS.InputPayload payload) {
List<TPM_RTRReportingWrapper_AMS.OutputRecord> output =
new List<TPM_RTRReportingWrapper_AMS.OutputRecord> ();
String pdim ;
String kdim ;
String tdim ;
Double v ;
AggregateResult[] groupedResults = [SELECT cgcloud__Anchor_Product__c,
sum (cgcloud__Deposits_Approved__c ) sumValue
from cgcloud__Fund__c
WHERE cgcloud__Anchor_Product__c IN :payload.productsfids
AND cgcloud__Anchor_Account__c IN :payload.accountsfids
AND ((cgcloud__Valid_From__c <= :inputDateEnd
AND cgcloud__Valid_From__c >= :inputDateBegin)
OR (cgcloud__Valid_Thru__c <= :inputDateEnd
AND cgcloud__Valid_Thru__c >= :inputDateBegin))
group by cgcloud__Anchor_Product__c
];
for(Integer i=0; i < groupedResults.size(); i++) {
v =(Double) groupedResults[i].get('sumValue');
pdim =(String) groupedResults[i].get('cgcloud__Anchor_Product__c');
kdim = 'myFndAmount';
tdim = 'Total';
output.add(new TPM_RTRReportingWrapper_AMS.OutputRecord(pdim ,kdim, tdim, v));
}
return output;
}
}
```

Use Case: Fixed Funds

In this use case, the custom datasource is used to include a fixed fund KPI in Real Time Report (RTR). As a result, the fixed funds are aggregated on both category and brand levels. One visualization option is this report, in which a brand value is only available for one brand. However, this value is not summed up to the category level. For the calculation of category levels, the data that's stored on Salesforce and Processing Services is used.

Prerequisites

1. Create a KPI definition for the fund amount. For this KPI definition, ensure that the object scope is Account and it has the `writeback` feature enabled.

Note: This KPI definition only provides a name for the external KPI. The Apex class defines how to enter values in this KPI.

2. Enter the name of the datasource table as `Account Measure`.
3. Add the KPI to the Reporting KPI set that's used in the report.

1. Create new KPI definitions for each use case, such as Fixed RDF Brand. Ensure that the object scope of these KPI definitions is Account.
2. Enter the name of the datasource table as `Account Measure`.
3. Define these KPI definitions in the report configuration
4. Create a custom metadata type, `TPM_RTRRouting__mdt`, which is linked to the KPI definition, sales organization, fund template, and KPI levels.

Field API Name	Type	Description
Fund_Template__c	Text	Determines the KPI to which the fixed funds are to be attributed. For example, All Transaction Row Records, in which the target fund has the RDF Brand Fund that belongs to the Fixed RDF KPI.
KPI_Definition__c	Text	Name of the Fixed Funds KPI (such as FXDR and FTDR) that determines the data which should be returned from the logic in the Apex class.
KPI_Level__c	Text	Fixed funds can be stored on different levels (currently, category and brand). This is needed to determine the method that should be run. (category and brand use different logic.)
Sales_Org__c	Text	Sales organization to which the record belongs.

5. On the `cgcloud__Fund_Transaction_Row__c` object, add a new field.

Field API Name	Type	Description	Formula
TPM_RTRAmount__c	Formula (Double)	Fixed Funds for Brands are calculated based on the value of the Amount field on the transaction record. Depending on the transaction type (withdraw or deposit), the sum of the aggregated amounts either increases (deposit) or decreases (withdraw). The formula field turns the amounts of records with transaction type = deposit to negative values. Instead of aggregating the amount field, you can aggregate the TPM_RTR_Amount__c field.	IF(ISPICKVAL(cgcloud__Transaction_Type__c , 'Withdraw'), cgcloud__Amount__c * (-1), cgcloud__Amount__c)

6. On the cgcloud__Fund_Transaction_Header__c object, add a new field.

Field API Name	Type	Description
TPM_Product__c	Lookup(Product)	Used to link products to the Multi-Fund Transaction object.

7. On the cgcloud__Fund_Template__c object, add a new field.

Field API Name	Type	Description
TPM_RTRRoutingFundType__c	Picklist	Used to filter fund template records. The picklist values of this field are fund template record names.

Note: Ensure that you create new fund templates for each case, such as RDF Brand Fund. Maintain the newly created TPM_RTRRoutingFundType__c field on the fund template.

Apex Classes

Custom data sources are related to two base Apex classes. Because all reports share these classes, you can use both classes only for code-routing, not to perform any actual logic, queries, or calculations.

Note: In RTR, the integration user runs both Apex classes, not the user who opens the report. As a result, the integration user must have access permissions for the used Apex classes.

TPM_RTRReportingWrapper_AMS (Base Class)

`TPM_RTRReportingWrapper_AMS` is a base class that contains the structures to load the input. This class contains the list of KPIs and filter criteria that you chose in the report—selected time frame or the list of customers and products. Depending on various use cases, the `payload` method can contain different attributes.

TPM_RTRSalesforceMonthlyMeasures_AMS (Base Class)

`TPM_RTRSalesforceMonthlyMeasures_AMS` is the second base class that's used for code-routing and is required to retrieve the information by a Salesforce REST endpoint `(/services/apexrest/RTRSalesforceMonthlyMeasures)`.

TPM_RTRFixedFunds_AMS (The Logic Class)

The actual logic is defined in this class. The benefit of routing to a separate third class is that the code runs only when needed.

TPM_RTRReportingWrapper_AMS (Base Class)

`TPM_RTRReportingWrapper_AMS` is a base class that contains the structures to load the input. This class contains the list of KPIs and filter criteria that you chose in the report—selected time frame or the list of customers and products. Depending on various use cases, the `payload` method can contain different attributes.

The OutputRecord method returns `product (pdim)`, `KPI (kdim)`, `time (tdim)`, and `values (v)`. The values are defined in the actual logic class, which is `TPM_RTRFunds_AMS` in this use case. For example, you can use tdim to display values on a more granular view—weeks, months, or quarters. If a report meta has monthly values, the external datasource can also provide monthly values.

Note: Ensure that `kdim` contains the name of the fund KPI.

Sample Code

```
global with sharing class TPM_RTRReportingWrapper_AMS
{
global class PeriodMonth {
global Integer year;
global Integer start;
global Integer total;
}
global class InputPayload{
global List <String> accountsfids;
global List <String> productsfids;
global PeriodMonth periodmonth;
global List <String> kpis;
global String responsetype;
global String callingsfuser;
}
global class OutputRecord{
global String pdim;
global String kdim;
global String tdim;
global Double v;
global OutputRecord( String pdim, String kdim , String tdim , Double v){
this.pdim = pdim;
this.kdim = kdim;
this.tdim = tdim;
this.v = v;
}
}
}
```

TPM_RTRSalesforceMonthlyMeasures_AMS (Base Class)

`TPM_RTRSalesforceMonthlyMeasures_AMS` is the second base class that's used for code-routing and is required to retrieve the information by a Salesforce REST endpoint `(/services/apexrest/RTRSalesforceMonthlyMeasures)`.

```
@RestResource(urlMapping='/RTRSalesforceMonthlyMeasures')
global with sharing class TPM_RTRSalesforceMonthlyMeasures_AMS {
@HttpGet
global static String doGet(){
return 'Hello world!';
}
@HttpPost
global static List<TPM_RTRReportingWrapper_AMS.OutputRecord> doPost(){
RestContext.response.statuscode = 200;
```

These are the code details:

1. The payload is received, and is then turned into an object of our wrapper class. The payload contains the KPIs that are defined in the report configuration.

```
String requestBody = RestContext.request.requestBody.toString();
TPM_RTRReportingWrapper_AMS.InputPayload payload =
(TPM_RTRReportingWrapper_AMS.InputPayload) JSON.deserialize(requestBody,
TPM_RTRReportingWrapper_AMS.InputPayload.class);
System.debug(payload);
if(payload.error == true) {
RestContext.response.statuscode = 500;
return null;
}
```

2. The integration user runs the base classes, not the user who opens the report. As a result, ensure that you query an account on the payload that the user selected to retrieve the actual user's sales org.

```
//Get Sales Org of the user who initiated the report
Account acc = [SELECT Id, cgcloud__Sales_Org__c FROM Account WHERE Id IN
:payload.accountsfids LIMIT 1];
String userSFOrg = acc.cgcloud__Sales_Org__c;
```

3. Helper maps and lists are created.

```
//Helper Maps/List
List<TPM_RTRReportingWrapper_AMS.OutputRecord> output = new
List<TPM_RTRReportingWrapper_AMS.OutputRecord>();
Map<String, TPM_RTRRouting__mdt> categoryKPIs = New Map<String, TPM_RTRRouting__mdt>();
Map<String, TPM_RTRRouting__mdt> brandKPIs = New Map<String, TPM_RTRRouting__mdt>();
```

4. The dates from the payload are converted to a workable format.

```
//Loading and transforming the dates selected in the filter
Date inputDateBegin = Date.newInstance(payload.periodmonth.year,
payload.periodmonth.start+1, 1);
Date inputDateEnd = inputDateBegin.addMonths(payload.periodmonth.total).addDays(-1);
```

5. The Apex class contains use case-specific code. Here, you can query all custom metadata records that were defined earlier and that belong to the sales org of the user. Then you can check whether the payload contains any KPIs in the custom metadata records. If it does, verify the level on which those KPIs are available and then add them to a dedicated map (category KPIs or brand KPIs map).

```
//Check against custom metadata if payload contains relevant KPIs and sort them
For(TPM_RTRRouting__mdt kpi : [SELECT Id, KPI_Definition__c, KPI_Level__c,
Fund_Template__c, Sales_Org__c FROM TPM_RTRRouting__mdt WHERE Sales_Org__c = :userSFOrg]){
If(payload.kpis.contains(kpi.KPI_Definition__c)){
If(kpi.KPI_Level__c == 'Category'){
categoryKPIs.put(kpi.KPI_Definition__c, kpi);
}
Else If(kpi.KPI_Level__c == 'Brand'){
brandKPIs.put(kpi.KPI_Definition__c, kpi);
}
}
}
```

6. You can check whether the two maps (brandKPIs and categoryKPIs) contain any values — if not, nothing happens. If they do, the actual class where all the logic happens is called and parameters such as the payload are passed.

```
//Run logic
try{
If(!categoryKPIs.isEmpty()){
TPM_RTRFixedFunds_AMS.getCategory(inputDateBegin, inputDateEnd, payload, output,
categoryKPIs);
}
If(!brandKPIs.isEmpty()){
TPM_RTRFixedFunds_AMS.getBrand(inputDateBegin, inputDateEnd, payload, output, brandKPIs);
}
}
catch(Exception ex){
RestContext.response.statuscode = 400;
System.debug(ex);
}
return output;
}
}
```

TPM_RTRFixedFunds_AMS (The Logic Class)

The actual logic is defined in this class. The benefit of routing to a separate third class is that the code runs only when needed.

For example, the Apex classes run only when the report contains the custom datasource and any of the KPIs defined in the custom metadata. If the report contains `fixed funds` KPIs only on the brand level, then only that particular method is run while the method that's related to categories is skipped. Also if the report contains fixed funds on brand level for only one particular template, then only this particular KPI is aggregated, calculated, and returned.

It's important to see how to write back values to the report. For the fixed funds logic, grouping, aggregations, and calculations are performed. And when that logic is performed, the values are written back to the report. This is done by adding the values to the output list and then returning it.

The `OutputRecord` method in the wrapper class creates the output list and contains product, KPI, time, and value.

Create Reports with Custom Apex Filter

Along with standard filters, you can also define custom Apex filters that reference the attributes of the Promotion object.

Core examples are filtering by the promotion type and phase of the promotion. A custom example is linking one promotion to one specific event.

With that, the main benefit of a custom Apex filter is to offer the opportunity to filter promotions by promotion attributes that aren't covered by standard filters. In addition to that, if all the values of the standard filter aren't visible, you can limit the available filter values of a promotion attribute for a specific real-time report.

Note: Custom Apex filters work only with single-select picklists, not with multi-select picklists.

Report Configuration

In the report configuration, ensure that you define the custom Apex filters in the same section as standard filters.

Note: You can't use a standard and custom filter with the same name (for example, promo_templatesfid) in the same report configuration.

Example: **Sample Code**

```
{
"type": "singleselect",
"label": "Promotion Type",
"name": "promo_templatesfid",
"source": {
"class": "TPM_RTRPromoTypeFilter_AMS",
"method": "Event"
},
"defaultValue": "<<FIRST_VALUE>>"
```

Here, `method` is the name of a parameter. This parameter is the `Developer_Name__c` field of a metadata type record.

However, you can also write a filter without parameters and discard the `method` line and modify the Apex code accordingly.

Apex Class

To implement the new filter logic, create an Apex class. You can construct the code for all Apex filters as specified here:

1. Implement `System.Callable`.
2. Retrieve the user's sales organization for later use.
3. Define the custom logic that returns a map with values.
4. Define a public object call that the report always initially calls. When the public object is called, it calls the method with the custom logic and passes a string parameter (based on the value of `method` in the report configuration).

Classes

Detailed samples of the classes that are used in report configurations.

TPM_RTRFixedFunds_AMS

Sample of the TPM_RTRFixedFunds_AMS class.

```
global class TPM_RTRFixedFunds_AMS{
global static List<TPM_RTRReportingWrapper_AMS.OutputRecord> getBrand(Date inputDateBegin,
 Date inputDateEnd, TPM_RTRReportingWrapper_AMS.InputPayload payload,
List<TPM_RTRReportingWrapper_AMS.OutputRecord> output, Map<String, TPM_RTRRouting__mdt>
fixedFunds){
```

```
//This code creates a list based on the map that was passed into the method. In this List
 only the Fund Template
//Names are stored so that the list can be used in the AggregateResult query (where clause).
 The goal is to only load records that
//are related to the Fund Templates we're looking at.
List<String> FundTemplateTypes = New List<String>();
String SalesOrg;
For(String cmID : fixedFunds.keyset()){
FundTemplateTypes.add(fixedFunds.get(cmID).Fund_Template__c);
if(Salesorg == Null){
Salesorg = fixedFunds.get(cmID).Sales_Org__c;
}
}
//The query returns Fund_Transaction_Row records based on the selected timeframe, customer,
 category and templates
//The results are aggregated TPM_RTR_Amount values grouped by BRAND, FUND TEMPLATE
AggregateResult[] groupedResults = [SELECT SUM(TPM_RTRAmount__c) amount,
cgcloud__Target_Fund__r.cgcloud__Fund_Template__r.Name template,
cgcloud__Target_Fund__r.cgcloud__Fund_Template__r.TPM_RTRRoutingFundType__c
RTRRoutingFundType,
cgcloud__Fund_Transaction__r.cgcloud__Fund_Transaction_Header__r.TPM_Product__c brand
FROM cgcloud__Fund_Transaction_Row__c
WHERE cgcloud__Target_Fund__r.cgcloud__Fund_Template__r.TPM_RTRRoutingFundType__c in
:FundTemplateTypes
AND cgcloud__Target_Fund__r.cgcloud__Sales_Org__c= :SalesOrg
AND
cgcloud__Fund_Transaction__r.cgcloud__Fund_Transaction_Header__r.TPM_Product__r.cgcloud__Criterion_1_Product__c
 IN :payload.productsfids
AND cgcloud__Target_Fund__r.cgcloud__Anchor_Account__c IN :payload.accountsfids
AND cgcloud__Transaction_Type__c IN ('Deposit','Withdraw')
AND ((cgcloud__Target_Fund__r.cgcloud__Valid_From__c <= :inputDateEnd
AND cgcloud__Target_Fund__r.cgcloud__Valid_From__c >= :inputDateBegin) OR
(cgcloud__Target_Fund__r.cgcloud__Valid_Thru__c <= :inputDateEnd
AND cgcloud__Target_Fund__r.cgcloud__Valid_Thru__c >= :inputDateBegin))
GROUP BY cgcloud__Fund_Transaction__r.cgcloud__Fund_Transaction_Header__r.TPM_Product__c,
 cgcloud__Target_Fund__r.cgcloud__Fund_Template__r.Name,
cgcloud__Target_Fund__r.cgcloud__Fund_Template__r.TPM_RTRRoutingFundType__c];
String brand, template, RTRRoutingFundType;
Double amount;
Map<String, Decimal> templateMap;
Map<String, Map<String, Decimal>> cmap = New Map<String, Map<String, Decimal>>();
Map<String, String> RTRRoutingFundTypeMap = New Map<String, String>();
//This For-Loop works on the aggregateResults and allocates all the records to their
respective "place" -
//it basically sorts all data into the right maps
For(AggregateResult ar : groupedResults){
brand = String.valueOf(ar.get('brand')); //the last part of this line retrieved the brand
 id of our query (brand is an alias)
template = String.valueOf(ar.get('template')); //the last part of this line retrieved the
 template Name of our query (template is an alias)
RTRRoutingFundType = String.valueOf(ar.get('RTRRoutingFundType')); //the last part of this
 line retrieved the RTR Routing Fund Type of our query (RTRRoutingFundType is an alias)'));
amount = Double.valueOf(ar.get('amount')); //the last part of this line retrieved the
aggregated amount of our query (amount is an alias)
```

```
templateMap = New Map<String, Decimal>();
If(!cmap.containsKey(brand)){
templateMap.put(template, amount);
cmap.put(brand, templateMap);
}
else {
templateMap = cmap.get(brand);
if(!templateMap.containsKey(template)){
templateMap.put(template, amount);
cmap.put(brand, templateMap);
}
}
if(!RTRRoutingFundTypeMap.containskey(RTRRoutingFundType)){
RTRRoutingFundTypeMap.put(RTRRoutingFundType, template);
}
}
//Once all the results of our query are allocated and sorted, the data needs to be written
 back to the report
//For each brand in cmap,
For(String brandId : cmap.keyset()){
//and each template for that brand,
For(String templateId : cmap.get(brandId).keyset()){
//Now we loop trough the map that was passed to this method
For(String ffID : fixedFunds.keyset()){
//to check to which KPI the map we're currently in is corrosponding to
If(templateId == RTRRoutingFundTypeMap.get(fixedFunds.get(ffID).Fund_Template__c)){
//If the map we're working on right now has the RDF Fund Template, and one of the records
 in our map has
//the RDF Fund Template assigned, we write back this KPI to the corrosponding KPI Definition
 of the Custom Meta Data
output.add(new TPM_RTRReportingWrapper_AMS.OutputRecord(String.valueOf(brandId),
String.valueOf(fixedFunds.get(ffID).KPI_Definition__c), 'Total',
Double.valueOf(cmap.get(brandId).get(templateId))));
}
}
}
}
return output;
}
global static List<TPM_RTRReportingWrapper_AMS.OutputRecord> getCategory(Date inputDateBegin,
 Date inputDateEnd, TPM_RTRReportingWrapper_AMS.InputPayload payload,
List<TPM_RTRReportingWrapper_AMS.OutputRecord> output, Map<String, TPM_RTRRouting__mdt>
fixedFunds){
//This code creates a list based on the map that was passed into the method. In this List
 only the Fund Template
//Names are stored so that the list can be used in the AggregateResult query. The goal is
 to only load records that
//are related to the Fund Templates we're looking at.
List<String> FundTemplateTypes = New List<String>();
String SalesOrg;
For(String cmID : fixedFunds.keyset()){
FundTemplateTypes.add(fixedFunds.get(cmID).Fund_Template__c);
if(SalesOrg == Null){
SalesOrg = fixedFunds.get(cmID).Sales_Org__c;
```

```
}
}
//Calculation of the current week number
Date todaysDate = date.today();
Date todaysDateInstance = date.newInstance(todaysdate.year(), todaysdate.month(),
todaysdate.day());
todaysdate.day();
Integer currentyear = todaysdate.year();
Date startDate = date.newInstance(currentyear, 1,1);
Integer numberDaysDue = startDate.daysBetween(todaysDateInstance);
Integer numberOfWeek = math.mod(Integer.valueOf(math.floor((numberDaysDue))/7),52)+1;
//Support Map for NA_REP_FixedFund_A Calculation
Map<String, Decimal> mPrdAmounts = new Map<String, Decimal>();
//Query
AggregateResult[] groupedResults = [SELECT SUM(cgcloud__Deposits_Approved__c) sumValue,
cgcloud__Anchor_Product__c,
cgcloud__Fund_Template__c,
cgcloud__Fund_Template__r.Name tmplName,
cgcloud__Fund_Template__r.TPM_RTRRoutingFundType__c RTRRoutingFundType
FROM cgcloud__Fund__c
WHERE cgcloud__Fund_Template__r.TPM_RTRRoutingFundType__c IN :FundTemplateTypes
AND cgcloud__Sales_Org__c= :SalesOrg
AND cgcloud__Anchor_Product__c IN :payload.productsfids
AND cgcloud__Anchor_Account__c IN :payload.accountsfids
AND ((cgcloud__Valid_From__c <= :inputDateEnd
AND cgcloud__Valid_From__c >= :inputDateBegin)
OR (cgcloud__Valid_Thru__c <= :inputDateEnd
AND cgcloud__Valid_Thru__c >= :inputDateBegin))
GROUP BY cgcloud__Anchor_Product__c, cgcloud__Fund_Template__c,
cgcloud__Fund_Template__r.Name, cgcloud__Fund_Template__r.TPM_RTRRoutingFundType__c];
For (AggregateResult ar : groupedResults){
//Check if current product is not already in the map, if so, add to the map with value 0
If(!mPrdAmounts.containsKey(String.valueOf(ar.get('cgcloud__Anchor_Product__c')))){
mPrdAmounts.put(String.valueOf(ar.get('cgcloud__Anchor_Product__c')), 0);
}
//Add the sum of the aggregateResult to the existing value in map for current product
mPrdAmounts.put(String.valueOf(ar.get('cgcloud__Anchor_Product__c')),
mPrdAmounts.get(String.valueOf(ar.get('cgcloud__Anchor_Product__c'))) +
Double.valueOf(ar.get('sumValue')));
//Check Fund Template and add KPI
For(String qKPI : fixedFunds.keyset()){
If(String.valueOf(ar.get('RTRRoutingFundType')) == fixedFunds.get(qKPI).Fund_Template__c){
output.add(new
TPM_RTRReportingWrapper_AMS.OutputRecord(String.valueOf(ar.get('cgcloud__Anchor_Product__c')),
 String.valueOf(fixedFunds.get(qKPI).KPI_Definition__c), 'Total',
Double.valueOf(ar.get('sumValue'))));
}
}
}
//Add NA_REP_FixedFund_A if required
If(fixedFunds.containsKey('NA_REP_FixedFund_A')){
For(String cdr : mPrdAmounts.keyset()){
output.add(new TPM_RTRReportingWrapper_AMS.OutputRecord(String.valueOf(cdr),
'NA_REP_FixedFund_A', 'Total', Double.valueOf((mPrdAmounts.get(cdr)*numberOfWeek/52))));
```

```
}
}
return output;
}
}
```

TPM_RTRPromoTypeFilter_AMS

Sample of the TPM_RTRPromoTypeFilter_AMS class.

```
public with sharing class TPM_RTRPromoTypeFilter_AMS implements System.Callable {
public class TPM_RTRPromotionsCallableException extends Exception {}
private static String userSalesOrgName = null;
//GET SF ID
static {
List<User> users = [SELECT cgcloud__Sales_Org__c FROM User WHERE Id =
:String.escapeSingleQuotes(UserInfo.getUserId()) LIMIT 1];
userSalesOrgName = users[0].cgcloud__Sales_Org__c;
}
@testVisible private static List<Map<String, Object>> getPromotionTypes(String devname) {
String myCM;
```

1. Query the custom metadata record based on the parameter, and then perform workarounds for the test classes.

   ```
   //GET CUSTOM METADATA TYPE PROMO TYPE GROUPINGS (BASED ON ARGUMENT OF REPORT)
   if(!Test.isRunningTest()){
   TPM_RTRPromoTypes__mdt customMetaData = [SELECT Promo_Types__c FROM TPM_RTRPromoTypes__mdt
    WHERE DeveloperName = :devname LIMIT 1];
   myCM = customMetaData.Promo_Types__c;
   }
   //WORKAROUND FOR TESTCLASS (CUSTOM METADATA TYPE)
   else {
   myCM = devname;
   }
   ```

2. From the custom metadata record, split the value of the `Promo_Types__c` field into individual values and then parse them into a new list type called groups.

   ```
   List<String> splitGrouping = myCM.split(';');
   List<String> typeGroups = New List<String>();
   For(String item : splitGrouping){
   typeGroups.add(item.trim());
   }
   ```

3. Create a new list that you can later use to store the value of `results = List<Map<String, Object>>`. You can also query promotion templates into another active list that has the same sales org as the user and whose name matches the values in the Groups list type.

   ```
   List<Map<String, Object>> result = new List<Map<String, Object>>();
   List<cgcloud__Promotion_Template__c> prmTmpl = New List<cgcloud__Promotion_Template__c>();
   prmTmpl = [SELECT Id, Name FROM cgcloud__Promotion_Template__c WHERE cgcloud__Active__c
    = true AND cgcloud__Sales_Org__c = :userSalesOrgName AND Name IN :typeGroups];
   Map<String, String> outP = New Map<String, String>();
   ```

4. Return the result.

```
for(cgcloud__Promotion_Template__c tmpl : prmTmpl){
If(!outP.containsKey(tmpl.Name)){
outP.put(String.valueOf(tmpl.Id), String.valueOf(tmpl.Name));
}
}
for(String typeId : outP.keyset()){
result.add(new Map<String, Object> {
'label' => outP.get(typeId),
'value' => typeId
});
}
return result;
}
public Object call(String method, Map<String, Object> args) {
return getPromotionTypes(method);
}
}
```

TPM_RTRReportingParentPromoFilter

Sample of the TPM_RTRReportingParentPromoFilter class.

```
public with sharing class TPM_RTRReportingParentPromoFilter implements System.Callable {
public class TPM_RTRPromotionsCallableException extends Exception {}
private static List<User> users = [SELECT cgcloud__Sales_Org__c FROM User WHERE Id =
:String.escapeSingleQuotes(UserInfo.getUserId()) LIMIT 1];
private static String userSalesOrgName = users[0].cgcloud__Sales_Org__c;
```

1. Create a new list that you can later use to store the value of `results = List<Map<String, Object>>`.

```
@TestVisible private static List<Map<String, Object>> getParentPromotion() {
List<Map<String, Object>> result = new List<Map<String, Object>>();
```

2. Query the Promotion object on all records templates that have the `CustomerEvents` picklist value selected on the `TPM_Promo_Type_ControlView__c` field for the appropriate sales org ad user.

```
List<cgcloud__Promotion__c> prmTmpl = [SELECT ID, Name, cgcloud__Slogan__c ,
cgcloud__Account__r.cgcloud__Account_Name__c, cgcloud__Promotion_Template__r.cgcloud__Description__c, cgcloud__Promotion_Template__r.cgcloud__Sales_Org__c, cgcloud__Promotion_Template__r.TPM_Promo_Type_ControlView__c
 FROM cgcloud__Promotion__c WHERE
cgcloud__Promotion_Template__r.TPM_Promo_Type_ControlView__c ='CustomerEvent' and
cgcloud__Promotion_Template__r.cgcloud__Sales_Org__c =: userSalesOrgName ];
Map<ID, String> outP = New Map<ID, String>();
for(cgcloud__Promotion__c templ : prmTmpl){
If(!outP.containsKey(templ.ID)){
outP.put(String.valueOf(templ.Id), String.valueOf(templ. cgcloud__Slogan__c));
}
}
```

3. Return the list.

```
for(String statusTo : outP.keyset()){
result.add(new Map<String, Object> {
```

```
'label' => outP.get(statusTo),
'value' => statusTo
});
}
system.debug(json.serialize(result));
return result;
}
public Object call(String method, Map<String, Object> args) {
return getParentPromotion();
}
}
```