Aggregator Guide
Oracle Banking APIs
Release 22.2.0.0.0

Part No. F72988-01

November 2022

ORACLE

Aggregator Guide

November 2022

Oracle Financial Services Software Limited

Oracle Park

Off Western Express Highway

Goregaon (East)

Mumbai, Maharashtra 400 063

India

Worldwide Inquiries:

Phone: +91 22 6718 3000

Fax:+91 22 6718 3001

www.oracle.com/financialservices/

Table of Contents

1. Preface

1.1 Intended Audience

This document is intended for the following audience:

- Customers
- Partners

1.2 Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

1.3 Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit

http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit

http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

1.4 Structure

This manual is organized into the following categories:

Preface gives information on the intended audience. It also describes the overall structure of the User Manual.

The subsequent chapters describes following details:

- Introduction
- Preferences & Database
- Configuration / Installation.

1.5 Related Information Sources

For more information on Oracle Banking APIs Release 22.2.0.0.0, refer to the following documents:

- Oracle Banking APIs Installation Manuals
- Oracle Banking APIs Licensing Guide

2. Aggregator Service

It is a generic service to list the required set of data like any normal List Service with the only difference that the data is in the most summarized view possible. This service is mostly used to draw graphical widgets on the screen that represents the summary of a particular domain.

2.1 Implementation

Returns the aggregated data requested for the given resource like for instance Account, for given data parameter like for instance Closing Balance, for given interval like for instance Daily, Monthly, for given period.

It has following parameters:

- resource - Resource for which the aggregation is required like Account. (Path Parameter)
- data - Data of the resource for which the aggregation is to be done like Closing balance for Account. (Query Parameter)
- grouping - Grouping for which the aggregation is required like type of transaction CREDIT or DEBIT. It is not mandatory to specify grouping. If not then default value will be “DEFAULT”. (Query Parameter)
- interval - The interval at which the aggregated data is required like Daily, Monthly. It is not mandatory to specify interval. If not then default value will be “D – Daily”. All available values are D - Daily, W - Weekly, F – Fortnightly, M - Monthly, Q - Quarterly, Y – Yearly. (Query Parameter)
- count - The number of times the aggregated data is required at given intervals. It is not mandatory to specify count. If not then default value will be 1. (Query Parameter)
- q - The generic filtering parameter. (Query Parameter)
- sortBy - sorting parameter to sort q param results. (Query Parameter)
- maxRecords - max records parameter to restrict count of q param results. (Query Parameter)

Now every combination of resource, data and grouping has its own service / implementation and an entry for the same is made in the seed.

This implementation is a service where all the business logic required to do aggregation is present including the actual call to get the original set of data before aggregation.

Now initially , the call goes to aggregator REST API with above mentioned parameters from which the control goes to aggregator service. Finally in aggregator service the fully qualified name of the actual service / implementation is fetched from the DB based on the combination of “resource”, “data” and “grouping” and actual aggregation is done.

For every service / implementation, an entry is made in DIGX_FW_CONFIG_ALL_B where prop_id is “resource.data.grouping” and category_id is “AggregatorConfig”. Now for one implementation there can be multiple grouping possible. In that case the prop_id will be “resource.data.grouping1, grouping2, ...groupingN”.

2.1.1 Example

Use case - User wants to summarize the total amount and the number of invoices raised for a particular program for supply chain finance. How can we acheive this ? For this purpose, you can use this aggregator service with the following parameters:

- **resource:** invoices
- **data:** Amount
- **grouping:** Program

Thus, in this case the prop_id will be “invoices.Amount.Program” where prop_value contains the fully qualified name of the service which represents the implementation for the same.

Similarly, if multiple grouping is to be done like based on program and currency, then the prop_id will be “invoice.Amount.Program,Currency” or “invoice.Amount.Currency,Program”.

Note : In case of multiple grouping, the order of comma separated grouping names used in prop_id and the ones sent in UI should be same.

i.e. In the above case, If the prop_id used is “invoice.Amount.Program,Currency” then from UI grouping should be sent in query parameter as “grouping=Program,Currency”.

****** In case of no grouping required then prop_id will look like “resource.data.DEFAULT” i.e. “invoice.Amount.DEFAULT”**

Now, the parameters like q, sortBy and maxRecords are the parameters for QQuery implementation which are used similarly the way they are used in other OBAPI services.

In case of aggregator service , these parameters will be directly passed to the implementation service where they can be used in actual call made to get the original data before aggregation.

2.2 Implementation Details of Individual Services

2.2.1 SCF Implementation

2.2.1.1 Case 1

We want the list of top programs between the logged in party and specified associated party for both roles of logged in party - buyer and supplier. To achieve this we fire the list of Invoices with below specified query parameters and then group them by program code (grouping invoices of same program) and finally calculate the sum of invoice amount for each group.

Prop_id: invoices.Amount.Program

URL: /digx-scf/v1/aggregator/invoices

Request Parameters:

data: Amount

grouping: Program

q:

1. Role of the logged in party that is used to get list of invoices(B or S) .
2. Invoice status - ACCEPTED, RAISED, FINANCED, PARTIAL_FINANCED to get outstanding invoices.
3. Payment status – UNPAID, PART_PAID, OVERDUE to get outstanding invoices.
4. Associated party id – To get list of invoices only linked between logged in party id and associated party id specified.

sortBy: We want only top programs so we sort the groups by total amount. Thus we send

amount and DESC(Descending).

maxRecords: We want only top 5 programs and thus we send 5.

Response:


```
{
   "aggregatedData": {
      "resource": "invoices",
      "groups": [{
         "id": "HPRF Program~HP Reverse Factoring~A~B",
         "intervals": [{
            "amount": {
               "currency": "GBP",
               "amount": 62739.98688000
            },
            "count": 4
         }]
      }, {
         "id": "HPPRGFIN1~HPPRGFIN1~A~B",
         "intervals": [{
            "amount": {
               "currency": "GBP",
               "amount": 8760.63785888
            },
            "count": 8
         }]
      }]
   }
}
```

Here, “groups” is the array which contains list of data after grouping. i.e. different groups. Inside group, “id” represents the unique Id of that group. In the response of this API, “id” will always be combination of program code, program name, relation of logged in party in that program (A or CP) and role of logged in party in that program (B or S), all tilde(~) separated.

Intervals contains the actual data (in this case, the total amount for each program) at various intervals. Here since no interval is specified there will always be one element in intervals array.

2.2.1.2 Case 2

We want the list of programs currency wise between the logged in party and specified associated party for both roles of logged in party - buyer and supplier. To achieve this we fire the list of Invoices with below specified query parameters and then group them by program code (grouping invoices of same program), then group them by currency(grouping invoices of same currency for every program) and finally calculate the sum of invoice amount for each group.

Prop_id: invoices.Amount.Currency,Program or invoices.Amount.Program,Currency

URL : /digx-scf/v1/aggregator/invoices

Request Parameters:

data: Amount

grouping: Currency,Program or Program,Currency

q:

1. Role of the logged in party that is used to get list of invoices(B or S) .
2. Invoice status - ACCEPTED, RAISED, FINANCED, PARTIAL_FINANCED to get outstanding invoices.
3. Payment status – UNPAID, PART_PAID, OVERDUE to get outstanding invoices.
4. Associated party id – To get list of invoices only linked between logged in party id and associated party id specified.

Response:


```
{
 "aggregatedData": {
 "resource": "invoices",
 "groups": [{
 "identifiers": ["GBP", "HPFactoringWRec~HP Factoring WRec~A~S"],
"intervals": [{
   "amount": {
      "currency": "GBP",
      "amount": 7426.00
   },
   "count": 37
}]
  }, {
   "identifiers": ["USD", "HPFactoringWRec~HP Factoring WRec~A~S"],
   "intervals": [{
      "amount": {
         "currency": "USD",
         "amount": 39840.00
      },
      "count": 295
   }]
  }]
 }
}
```

Here, “groups” is the array which contains list of data after grouping. i.e. different groups. Inside group, “identifiers” represents the unique Id of that group. Since, multiple grouping is done, id is the list containing 2 elements. First is Currency and second Is the combination of program code, program name, relation of logged in party in that program (A or CP) and role of logged in party in that program (B or S), all tilde(~) separated.

Intervals contains the actual data (in this case, the total amount for each program, currency wise) at various intervals. Here since no interval is specified there will always be one element in intervals array.

2.2.1.3 Case 3

We want the list of top associated parties linked with logged in party in a particular program. To achieve this we fire the list of Invoices with below specified query parameters and then group them by associated party Id (grouping invoices of same associated party) and finally calculate the sum of invoice amount for each group.

Prop_id: invoices.Amount.AssociatedParty

URL : /digx-scf/v1/aggregator/invoices

Request Parameters:

data: Amount

grouping: AssociatedParty

q:

1. Role of the logged in party that is used to get list of invoices(B or S) .
2. Invoice status - ACCEPTED, RAISED, FINANCED, PARTIAL_FINANCED to get outstanding invoices.
3. Payment status – UNPAID, PART_PAID, OVERDUE to get outstanding invoices.
4. program code – To get list of invoices only linked between logged in party id and associated party id in the specified program.

sortBy: We want only top associated parties so we sort the groups by total amount. Thus we send amount and DESC(Descending).

maxRecords: We want only top 10 associated parties and thus we send 10.

Response:


```
{
"aggregatedData": {
"resource": "invoices",
"groups": [{
"id": " E4228ED58341003545623EDC7319024990E5C38ACB60 ~***728 ~TURBO TEXTILES ",
"intervals": [{
"amount": {
"currency": "GBP",
"amount": 7426.00
},
"count": 37
}]
}, {
"id": " E4228ED58341003545623EDC7319024990E5C38ACB90~*****C001 ~TestCP02 "],
"intervals": [{
"amount": {
"currency": "USD",
"amount": 39840.00
},
"count": 295
}]
}]
}
}
```

Here, “groups” is the array which contains list of data after grouping. i.e. different groups. Inside group, “id” represents the unique Id of that group. In the response of this API, “id” will always be combination of associated party id (hashed value), associated party id (display value), all tilde(~) separated.

Intervals contains the actual data (in this case, the total amount for each associated party) at various intervals. Here since no interval is specified there will always be one element in intervals array.

2.2.1.4 Case 4

We want the list of associated parties currency wise linked with logged in party in a particular program. To achieve this we fire the list of with below specified query parameters and then group them by associated party id (grouping invoices of same associated party), then group them by currency(grouping invoices of same currency for every associated party) and finally calculate the sum of invoice amount for each group.

Prop_id: invoices.Amount.AssociatedParty,Currency or
invoices.Amount.Currency,AssociatedParty

URL: /digx-scf/v1/aggregator/invoices

Request Parameters:

data: Amount

grouping: AssociatedParty,Currency or Currency,AssociatedParty

q:

1. Role of the logged in party that is used to get list of invoices(B or S) .
2. Invoice status - ACCEPTED, RAISED, FINANCED, PARTIAL_FINANCED to get outstanding invoices.
3. Payment status – UNPAID, PART_PAID, OVERDUE to get outstanding invoices.
4. program code – To get list of invoices only linked between logged in party id and associated party id in the specified program.

Response:


```
          {
          "aggregatedData": {
          "resource": "invoices",
          "groups": [{
"identifiers": ["GBP", " E4228ED58341003545623EDC7319024990E5C38ACB60 ~***728
~TURBO TEXTILES "],
          "intervals": [{
            "amount": {
               "currency": "GBP",
               "amount": 7426.00
            },
            "count": 37
          }]
       }, {
"identifiers": ["USD", " E4228ED58341003545623EDC7319024990E5C38ACB60 ~***728
~TURBO TEXTILES "],
            "intervals": [{
              "amount": {
                 "currency": "USD",
                 "amount": 39840.00
              },
              "count": 295
            }]
       }]
   }
}
```

Here, “groups” is the array which contains list of data after grouping. i.e. different groups. Inside group, “identifiers” represents the unique Id of that group. Since, multiple grouping is done, id is the list containing 2 elements. First is Currency and second Is the combination of of associated party id (hashed value), associated party id (display value), all tilde(~) separated.

Intervals contains the actual data (in this case, the total amount for each associated party, currency wise) at various intervals. Here since no interval is specified there will always be one element in intervals array.

2.2.2 VAM Implementation

2.2.2.1 Case 1

Fetching list of value dated balances for the top N virtual accounts with respect to available balance for a selected virtual entity and currency. It fetched the

Prop_id: virtualAccounts.valueDated.DEFAULT

URL: GET + /digx-vam/v1/aggregator/resource/virtualAccounts

Request Parameters:

data: valueDated

maxRecords: 5 (Integer for number of virtual accounts)

q:

- virtualEntityId - the selected virtual entity id filter on virtual accounts
- vStatus – only open virtual accounts to be fetched
- availableBalance.currency – the selected currency filter on virtual accounts

sortParams:

- sortBy : availableBalance.amount
- sortOrder: DESC (Fetches top N)

Response:


```
          {
  "aggregatedData": {
    "resource": "virtualAccounts",
    "groups": [{
      "id": {
        "displayValue": "xxxxxxxxxxxx0096",
        "value": "C56C880F40EA1F354870342328EED1323799A835BE1813AA"
      },
      "intervals": [{
        "amount": {
          "currency": "GBP",
          "amount": -165
        },
        "date": "2018-10-02T00:00:00"
      }]
    }]
  }
}
```


Id - is the virtual account number
each group in groups array represents the balance for the value date in the group.

2.2.2.2 Case 2

Fetching list of virtual accounts for a selected virtual entity and group the virtual accounts based on the currency and aggregate the availableBalance of the virtual account to provide currency wise distribution to the user.

Prop_id: virtualAccounts.availableBalance.DEFAULT

URL: GET + /digx-vam/v1/aggregator/resource/virtualAccounts

Request Parameters:

data: availableBalance

q:

- virtualEntityId - the selected virtual entity id filter on virtual accounts
- vStatus – only open virtual accounts to be fetched

sortParams:

- sortBy : availableBalance.amount
- sortOrder: DESC (Fetches top N)

Response:


```
{
   "aggregatedData": {
      "resource": "virtualAccounts",
      "groups": [{
         "id": "EUR",
         "intervals": [{
            "amount": {
               "currency": "EUR",
               "amount": 1329
            },
            "count": 2
         }]
      }]
   }
}
```


Each group in the groups array represents the currency and its sum for the virtual accounts satisfying the criteria in the request and the number of virtual accounts in that criteria.

2.2.3 Receivables Implementation

2.2.3.1 Purchase Order

2.2.3.1.1 Case 1

We want the list of Purchase Orders status wise linked with logged in party. To achieve this we fire the list of purchase orders with below specified query parameters and then group them by status (grouping purchase orders of same status) and finally calculate the sum of purchase order amount for each group.

Prop_id: purchaseorders.Amount.Status

URL: /digx-scf/v1/aggregator/purchaseorders

Request Parameters:

data: Amount

grouping: Status

q: 1. Role of the logged in party that is used to get list of purchase orders(B or S) .

Response:


```
{

        "aggregatedData": {

                "resource": "purchaseorders",
```

```
"groups": [{
        "identifiers": ["ACCEPTED"],
        "intervals": [{
            "amount": {
                "currency": "EUR",
                "amount":
132708027.41077000910043978847419055000500520691275596618652343 75
            },
            "count": 70
        }]
    }, {
        "identifiers": ["RAISED"],
        "intervals": [{
            "amount": {
                "currency": "EUR",
                "amount":
4071865.52125000032070229472047628860309487208724021911621093 75
            },
            "count": 61
        }]
    }, {
        "identifiers": ["REJECTED"],
        "intervals": [{
            "amount": {
                "currency": "EUR",
                "amount":
20205.57000000000163146829379456903552636504173278808593 75
            },
            "count": 7
```

```
                }]

            }, {

                "identifiers": ["CANCELLED"],

                "intervals": [{

                    "amount": {

                        "currency": "EUR",

                        "amount":
1653850.000000000133537625401913828682154417037963867187500

                    },

                    "count": 4

                }]

            }]

        }

}
```


Here, “groups” is the array which contains list of data after grouping. i.e. different groups. Inside group, “identifiers” represents the unique Id of that group. Since, grouping is done on the basis of purchase order status, id contains purchase order’s status.

Intervals contains the actual data (in this case, the total amount for each status wise) at various intervals. Here since no interval is specified there will always be one element in intervals array.

2.2.3.1.2 Case 2

We want the list of Top 10 associated parties purchase order status wise linked with logged in party. To achieve this we fire the list of purchase orders with below specified query parameters and then group them by associated party id (grouping purchase orders of same associated party), then group them by purchase order status(grouping purchase orders of same status for every associated party) and finally calculate the sum of purchase order amount for each group.

Prop_id: purchaseorders.Amount.AssociatedParty,Status

URL: /digx-scf/v1/aggregator/purchaseorders

Request Parameters:

data: Amount

grouping: AssociatedParty,Status

q: 1. Role of the logged in party that is used to get list of purchase orders(B or S) .

2. Invoice status - ACCEPTED, RAISED.

Response:

{

"aggregatedData": {

"resource": "purchaseorders",

"groups": [{

"identifiers": ["98DCBD13A0F3EA4F5EDE19325B4CD2D30A1C949B838D0E171B1B~***000153~LINKINVA 23Dec", "ACCEPTED"],

"intervals": [{

"amount": {

"currency": "EUR",

"amount": 109765554.800000008862854272706499614287167787551879882812500

},

"count": 1

}]

}, {

"identifiers": ["E6AFBD17A6F1717F4C5BCEE75406C9BB37C8DF99577A~***462~ABZ Solutions", "ACCEPTED"],

"intervals": [{

"amount": {

"currency": "EUR",

"amount": 22904335.720770000234506210068374798538570757955312728881835937 5

},

"count": 62

}]

}, {

"identifiers": ["E6AFBD17A6F1717F4C5BCEE75406C9BB37C8DF99577A~***462~ABZ Solutions", "RAISED"],

```
                "intervals": [{
                    "amount": {
                        "currency": "EUR",
                        "amount":
4066573.157250000320274970766476485550811048597097396850585937 5
                    },
                    "count": 45
                }]
            }, {
                "identifiers":
["98DCBD13A0F3EB4D5ADBFB7E68621176D18D733CA71A0B515B69~***000077~Septonepar
ty", "ACCEPTED"],
                "intervals": [{
                    "amount": {
                        "currency": "EUR",
                        "amount":
22000.000000000001776356839400250464677810668945312500000
                    },
                    "count": 1
                }]
            }, {
                "identifiers":
["E6AFBC14A1F57BA066463EE6C10B5E5D5324274D7899~***716~AugBuyer", "ACCEPTED"],
                "intervals": [{
                    "amount": {
                        "currency": "EUR",
                        "amount":
10636.89000000000085885965006582409841939806938171386718 75
                    },
                    "count": 5
                }]
```

```
                }, {
                        "identifiers":
["98DCBD13A0F3EB4C5BD1361685D7E30CC39EA19106076A3CB8DB~***000066~LinkInvBuy
Cp", "ACCEPTED"],
                        "intervals": [{
                                "amount": {
                                        "currency": "EUR",
                                        "amount":
5500.000000000000444089209850062616169452667236328125000
                                },
                                "count": 1
                        }]
                }, {
                        "identifiers":
["E6AFBC14A1F57BA066463EE6C10B5E5D5324274D7899~***716~AugBuyer", "RAISED"],
                        "intervals": [{
                                "amount": {
                                        "currency": "EUR",
                                        "amount":
5292.364000000000427323953999803052283823490142822265625000
                                },
                                "count": 16
                        }]
                }]
        }
}
```


Here, “groups” is the array which contains list of data after grouping. i.e. different groups. Inside group, “identifiers” represents the unique Id of that group. Since, multiple grouping is done, id is the list containing 2 elements. First is the combination of associated party id (hashed value), associated party id (display value) and associated party name, all tilde(~) separated and second Is the purchase order status.

Intervals contains the actual data (in this case, the total amount for each associated party, status wise) at various intervals. Here since no interval is specified there will always be one element in intervals array.

2.2.3.2 Reconciliation

2.2.3.2.1 Case 1

We want the list of unmatched payments currency wise for the logged in party. To achieve this we fire the list of payments with below specified query parameters and then group them by currency (grouping payments of same currency), then group them by payment type(grouping payments of same type for every currency) and finally calculate the sum of payment amount for each group.

Prop_id: payments.Amount.Currency,Type

URL: /digx-cms/v1/aggregator/payments

Request Parameters:

data: Amount

grouping: Currency,Type

q: 1. Payment Status - UNMATCHED .

Response:


```
{
    "aggregatedData": {
        "resource": "payments",
        "groups": [{
            "identifiers": ["GBP", "C"],
            "intervals": [{
                "amount": {
                    "currency": "GBP",
                    "amount": 835619
                },
                "count": 7
            }]
        }, {
```

```
            "identifiers": ["USD", "D"],
            "intervals": [{
                "amount": {
                    "currency": "USD",
                    "amount": 346103
                },
                "count": 8
            }]
        }, {
            "identifiers": ["GBP", "D"],
            "intervals": [{
                "amount": {
                    "currency": "GBP",
                    "amount": 333903
                },
                "count": 7
            }]
        }, {
            "identifiers": ["USD", "C"],
            "intervals": [{
                "amount": {
                    "currency": "USD",
                    "amount": 40200
                },
                "count": 5
            }]
        }]
}
```

}

Here, “groups” is the array which contains list of data after grouping. i.e. different groups. Inside group, “identifiers” represents the unique Id of that group. Since, multiple grouping is done, id is the list containing 2 elements. First is the currency and second is the payment type.

Intervals contains the actual data (in this case, the total amount for each currency, payment type wise) at various intervals. Here since no interval is specified there will always be one element in intervals array.

2.2.3.2.2 Case 2

We want the list of unreconciled invoices currency wise for the logged in party. To achieve this we fire the list of invoices with below specified query parameters and then group them by currency (grouping invoices of same currency), then group them by role wise(grouping invoices of same role for every currency) and finally calculate the sum of invoice amount for each group.

Prop_id: invoices.Amount.Currency,Role

URL: /digx-invoice/v1/aggregator/invoices

Request Parameters:

data: Amount

grouping: Currency,Role

q: 1. Payment Status - UNPAID .

Response :


```
{
    "aggregatedData": {
        "resource": "invoices",
        "groups": [{
            "identifiers": ["GBP", "B"],
            "intervals": [{
                "amount": {
                    "currency": "GBP",
                    "amount": 1047165452.0
                },
                "count": 281
```

```
        }]
    }, {
        "identifiers": ["GBP", "S"],
        "intervals": [{
            "amount": {
                "currency": "GBP",
                "amount": 4742388.0
            },
            "count": 75
        }]
    }, {
        "identifiers": ["USD", "B"],
        "intervals": [{
            "amount": {
                "currency": "USD",
                "amount": 17900.0
            },
            "count": 5
        }]
    }, {
        "identifiers": ["USD", "S"],
        "intervals": [{
            "amount": {
                "currency": "USD",
                "amount": 5000.0
            },
            "count": 3
        }]
```

```
}, {
        "identifiers": ["INR", "S"],
        "intervals": [{
                "amount": {
                        "currency": "INR",
                        "amount": 5000.0
                },
                "count": 1
        }]
}, {
        "identifiers": ["LAK", "B"],
        "intervals": [{
                "amount": {
                        "currency": "LAK",
                        "amount": 4401
                },
                "count": 37
        }]
}, {
        "identifiers": ["LAK", "S"],
        "intervals": [{
                "amount": {
                        "currency": "LAK",
                        "amount": 2024
                },
                "count": 7
        }]
}]
```

```
        }
}
```

Here, “groups” is the array which contains list of data after grouping. i.e. different groups. Inside group, “identifiers” represents the unique Id of that group. Since, multiple grouping is done, id is the list containing 2 elements. First is the currency and second is the associated party role.

Intervals contains the actual data (in this case, the total amount for each currency, associated party role wise) at various intervals. Here since no interval is specified there will always be one element in intervals array.

2.2.3.2.3 Case 3

We want the list of unreconciled cashflows currency wise for the logged in party. To achieve this we fire the list of cashflows with below specified query parameters and then group them by currency (grouping cashflows of same currency), then group them by cashflow type(grouping cashflows of same type for every currency) and finally calculate the sum of cashflow amount for each group.

Prop_id: cashflows.Amount.Currency,Type

URL: /digx-cms/v1/aggregator/cashflows

Request Parameters:

data: Amount

grouping: Currency,Type

q: 1. Reconciliation Status - UNRECONCILED .

Response :

```
{
        "aggregatedData": {
                "resource": "cashflows",
                "groups": [{
                        "identifiers": ["LAK", "I"],
                        "intervals": [{
                                "amount": {
                                        "currency": "LAK",
                                        "amount": 573993369
                                },
```

```
                "count": 82
            }]
        }, {
            "identifiers": ["GBP", "I"],
            "intervals": [{
                "amount": {
                    "currency": "GBP",
                    "amount": 59712749
                },
                "count": 194
            }]
        }, {
            "identifiers": ["USD", "I"],
            "intervals": [{
                "amount": {
                    "currency": "USD",
                    "amount": 36663338
                },
                "count": 177
            }]
        }, {
            "identifiers": ["GBP", "O"],
            "intervals": [{
                "amount": {
                    "currency": "GBP",
                    "amount": 1666225
                },
                "count": 67
```

```
        }]
    }, {
        "identifiers": ["LBP", "I"],
        "intervals": [{
            "amount": {
                "currency": "LBP",
                "amount": 991545
            },
            "count": 4
        }]
    }, {
        "identifiers": ["USD", "O"],
        "intervals": [{
            "amount": {
                "currency": "USD",
                "amount": 51044.33
            },
            "count": 264
        }]
    }, {
        "identifiers": ["EUR", "I"],
        "intervals": [{
            "amount": {
                "currency": "EUR",
                "amount": 28547.17
            },
            "count": 34
        }]
```

```
        }, {
                "identifiers": ["LAK", "O"],
                "intervals": [{
                        "amount": {
                                "currency": "LAK",
                                "amount": 4567
                        },
                        "count": 2
                }]
        }]
    }
}
```


Here, “groups” is the array which contains list of data after grouping. i.e. different groups. Inside group, “identifiers” represents the unique Id of that group. Since, multiple grouping is done, id is the list containing 2 elements. First is the currency and second is the cashflow type.

Intervals contains the actual data (in this case, the total amount for each currency, cashflow type wise) at various intervals. Here since no interval is specified there will always be one element in intervals array.

2.2.3.2.4 Case 4

We want the list of payments status wise for the logged in party. To achieve this we fire the list of payments with below specified query parameters and then group them by status (grouping payments of same status), then group them by type(grouping payments of same type for every status) and finally calculate the sum of payment amount for each group.

Prop_id: payments.Amount.Status,Type

URL: /digx-cms/v1/aggregator/payments

Request Parameters:

data: Amount

grouping: Status,Type

q: 1. Payment Date – from date and to date .

Response :

```
{
    "aggregatedData": {
        "resource": "payments",
        "groups": [{
            "identifiers": ["MATCHED", "C"],
            "intervals": [{
                "amount": {
                    "currency": "USD",
                    "amount": 291300
                },
                "count": 57
            }]
        }, {
            "identifiers": ["PART_MATCHED", "C"],
            "intervals": [{
                "amount": {
                    "currency": "GBP",
                    "amount": 9700
                },
                "count": 1
            }]
        }, {
            "identifiers": ["PART_MATCHED", "D"],
            "intervals": [{
                "amount": {
                    "currency": "GBP",
                    "amount": 32900
                },
```

```
                "count": 6
            }]
        }, {
            "identifiers": ["RECON_NA", "D"],
            "intervals": [{
                "amount": {
                    "currency": "USD",
                    "amount": 5200
                },
                "count": 1
            }]
        }, {
            "identifiers": ["MATCHED", "D"],
            "intervals": [{
                "amount": {
                    "currency": "GBP",
                    "amount": 580900
                },
                "count": 41
            }]
        }]
    }
}
```


Here, “groups” is the array which contains list of data after grouping. i.e. different groups. Inside group, “identifiers” represents the unique Id of that group. Since, multiple grouping is done, id is the list containing 2 elements. First is the payment status and second is the payment type.

Intervals contains the actual data (in this case, the total amount for each status, payment type wise) at various intervals. Here since no interval is specified there will always be one element in intervals array.

2.2.3.2.5 Case 5

We want the list of unmatched payments entity wise for the logged in party. To achieve this we fire the list of payments with below specified query parameters and then group them by payment entity (grouping payments of same entity), then group them by payment type(grouping payments of same type for every entity) and finally calculate the sum of payment amount for each group.

Prop_id: payments.Amount.Entity,Type

URL: /digx-cms/v1/aggregator/payments

Request Parameters:

data: Amount

grouping: Entity,Type

q: 1. Payment Status - UNMATCHED .

Response :


```
{
    "aggregatedData": {
        "resource": "payments",
        "groups": [{
            "identifiers": ["E", "C"],
            "intervals": [{
                "amount": {
                    "currency": "GBP",
                    "amount": 140412
                },
                "count": 5
            }]
        }, {
            "identifiers": ["I", "C"],
            "intervals": [{
                "amount": {
                    "currency": "USD",
```

```
                    "amount": 735407
                },
                "count": 7
            }]
        }, {
            "identifiers": ["I", "D"],
            "intervals": [{
                "amount": {
                    "currency": "USD",
                    "amount": 553005
                },
                "count": 9
            }]
        }, {
            "identifiers": ["E", "D"],
            "intervals": [{
                "amount": {
                    "currency": "GBP",
                    "amount": 127001
                },
                "count": 6
            }]
        }]
    }
}
```


Here, “groups” is the array which contains list of data after grouping. i.e. different groups. Inside group, “identifiers” represents the unique Id of that group. Since, multiple grouping is done, id is the list containing 2 elements. First is the payment entity and second is the payment type.

Intervals contains the actual data (in this case, the total amount for each entity, payment type wise) at various intervals. Here since no interval is specified there will always be one element in intervals array.

2.2.3.2.6 Case 6

We want the list of payments allocation status wise for the logged in party. To achieve this we fire the list of payments with below specified query parameters and then group them by allocation status (grouping payments of same allocation status), then group them by payment type wise(grouping payments of same type for every allocation status) .

Prop_id: payments.Count.AllocationStatus,Type

URL: /digx-cms/v1/aggregator/payments

Request Parameters:

data: Count

grouping: AllocationStatus,Type

q: 1. Payment Date – from date and to date .

Response :


```
{
    "aggregatedData": {
        "resource": "payments",
        "groups": [{
            "identifiers": ["FAILED", "C"],
            "intervals": [{
                "count": 3
            }]
        }, {
            "identifiers": ["UNALLOCATED", "C"],
            "intervals": [{
                "count": 55
            }]
        }, {
```

```
                "identifiers": ["UNALLOCATED", "D"],
                "intervals": [{
                        "count": 47
                }]
        }, {
                "identifiers": ["FAILED", "D"],
                "intervals": [{
                        "count": 1
                }]
        }]
    }
}
```


Here, “groups” is the array which contains list of data after grouping. i.e. different groups. Inside group, “identifiers” represents the unique Id of that group. Since, multiple grouping is done, id is the list containing 2 elements. First is the allocation status and second is the payment type.

Intervals contains the actual data (in this case, the total count for each allocation status, payment type wise) at various intervals. Here since no interval is specified there will always be one element in intervals array.

2.2.3.2.7 Case 7

We want the list of allocated transactions allocation type wise for the logged in party. To achieve this we fire the list of payments with below specified query parameters and then group them by allocation type (grouping payments of same allocation type), then group them by payment type(grouping payments of same type for every allocation type) .

Prop_id: allocations.Count.PaymentType,Type

URL: /digx-cms/v1/aggregator/allocations

Request Parameters:

data: Count

grouping: PaymentType,Type

Response :


```
{
```

```
        "aggregatedData": {
                "resource": "allocations",
                "groups": [{
                        "identifiers": ["C", "M"],
                        "intervals": [{
                                "count": 102
                        }]
                }, {
                        "identifiers": ["D", "M"],
                        "intervals": [{
                                "count": 50
                        }]
                }]
        }
}
```


Here, “groups” is the array which contains list of data after grouping. i.e. different groups. Inside group, “identifiers” represents the unique Id of that group. Since, multiple grouping is done, id is the list containing 2 elements. First is the payment type and second is the allocation type.

Intervals contains the actual data (in this case, the total count for each payment type, allocation type wise) at various intervals. Here since no interval is specified there will always be one element in intervals array.

Home