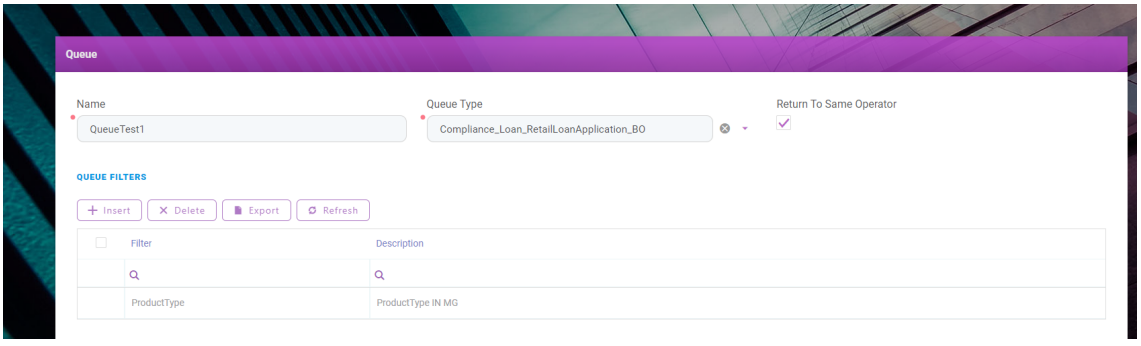
FTOS_CMB_TaskManagementHelper - server library

The server library “FTOS_CMB_TaskManagementHelper“ is part of the TaskManagement Module SDK. The following functions were implemented

1. `getAvailableQueue(filtersParam, queueTypeId, taskTypeId)`
   - ***Input***:
     - `filtersParam` - filters for queues
     - `queueTypeId` - queue type id (based on this or taskTypeId the initial list of queues is retrieved - at least one of these two params needs to be given)
     - `taskTypeId` - task type id (based on this or queueTypeId the initial list of queues is retrieved - at least one of these two params needs to be given)
   - ***Output***:
     - only one queue that is matching the parameters, in case two queues are found, an error is logged that the filters needs to be checked so the queue can be uniquely identified
   - ***CallExample:***

```
var filtersParams = [];
var productTypeFilter = {};
productTypeFilter.name = "ProductType";
productTypeFilter.value = "MG";
filtersParams.push(productTypeFilter);
var queueTypeId = 'e398ff08-9127-482d-b126-3758bb42b08d';
var taskTypeId = 'a198ff08-9127-482d-b126-3758bb42b08d';

//with queueTypeId
var resultedQueue = getAvailableQueue(filtersParam,
queueTypeId);

//or with taskTypeId
var resultedQueue = getAvailableQueue(filtersParam, null,
taskTypeId);

//or with both
var resultedQueue = getAvailableQueue(filtersParam,
queueTypeId, taskTypeId);
```

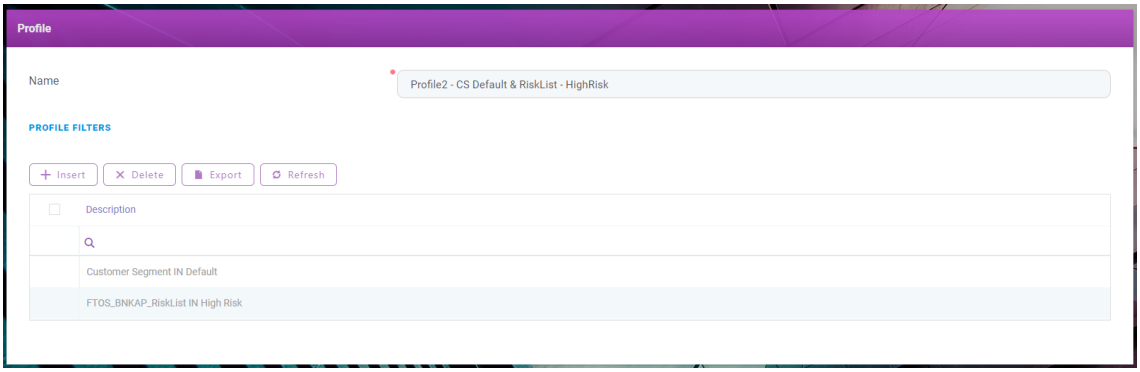
   - ***Technical description:***
     - on first step, all queues with the specific queueTypeId or taskTypeId are retrieved
     - filter all the queues from the above, meaning comparing the filters received with the filters added on each queue. In case some queues have no filters set (even if we receive filters param) those queues are automatically added on list of result.
     - in case two queues are found, an error is logged that the filters needs to be checked so the queue can be uniquely identified



2. `filterProfiles(profileFiltersParam)`
    - ***Input***: filters for profiles
    - ***Output***: only one profile that is matching the profile filter, in case two profiles are found, an error is logged that the filters needs to be checked so the profile can be uniquely identified
    - ***CallExample***:

```
var filtersParams = [];
var customerSegmentFilter = {};
customerSegmentFilter.name = "Customer Segment";
customerSegmentFilter.value = "Default";
filtersParams.push(productTypeFilter);
var skillFilter = {};
skillFilter.name = "Skill";
skillFilter.value = "eb44ba02-7428-4c60-9fa6-7e850435205b";
filtersParams.push(skillFilter);
//filters that are lookup, the value needs to be the ids;
//for option sets the value will be the name of the option set
item

var resultedProfile = filterProfiles(filtersParam);
```

    - ***Technical description:***
        - filter all the profiles, comparing the filters received with the filters added on each profile. In case some profiles have no filters set (even if we receive filters param) those profiles are automatically added on list of result.
        - in case two profiles are found, an error is logged that the filters needs to be checked so the profile can be uniquely identified



3. `getAvailableOperator(profileId, queueId, competenceLevelId, queueItemIsInReview)`
    - ***Input***:
        - `profileId`-profile id used for getting only operators with this profile id
        - `queueId` - queue id used for getting only operators on this queue
        - `competenceLevelId` - competence level id used for getting only operators with this competence level
        - `queueItemIsInReview` - if this is true, condition with `MaxActiveItemAllocation` should not be applied, meaning it does not matter if for that operator the `MaxActiveItemAllocation` has been reached
    - ***Output***: one operator for which a bunch of condition were satisfied
    - ***CallExample***:

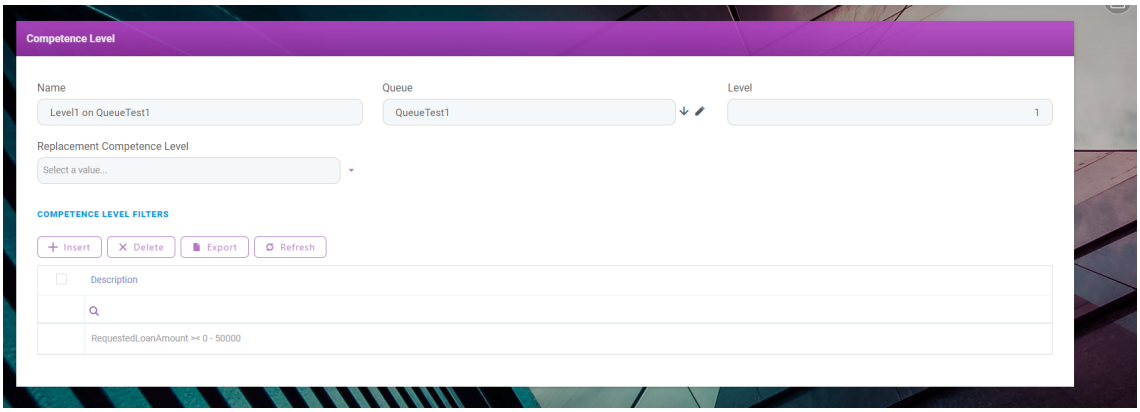
```
var profileId = 'eb44ba02-7428-4c60-9fa6-7e850435205b';
var queueId = 'ab44ba02-7428-4c60-9fa6-7e850435205b';
var competenceLevelId = 'bb44ba02-7428-4c60-9fa6-7e850435205b';
var availableOperator = getAvailableOperator(profileId ,
queueId ,competenceLevelId);

//or with param queueItemIsInReview
var availableOperator = getAvailableOperator(profileId ,
queueId ,competenceLevelId, true);
```

- ***Technical description:***
  - get no of allocated item on current queue - this number will be later on used on filtering the profiles
  - get operators based on other conditions: noOfAllocatedItems, queueId, operator status, competence level and ordered ascending by lastAllocationDate and ItemCount
    - if queue item is in review, condition with `MaxActiveItemAllocation` should not be applied, meaning it does not matter if for that operator the `MaxActiveItemAllocation` has been reached
  - first operator from the above list will be returned

4. `getCompetenceLevels(filtersParam, queueId)`
   - ***Input***:
     - `filtersParam` - filters for competence level
     - `queueId` - queue id - this will be used for getting only the competence levels for this queue
   - ***Output***: a list of competence levels which will satisfy the filters and queueid param
     - a competence level object will contain the following attributes:
       - itemId the id of competence level
       - itemName name of competence level
       - level level no of the competence level
   - ***CallExample***:

```
var filtersParams = [];
var requestedLoanAmountFilter = {};
requestedLoanAmountFilter.name = "RequestedLoanAmount";
requestedLoanAmountFilter.value = "57000";
filtersParams.push(requestedLoanAmountFilter);
var queueId = 'e398ff08-9127-482d-b126-3758bb42b08d';

var competenceLevels = getCompetenceLevel(filtersParam,
queueId );
```

   - ***Technical description:***
     - get competence levels that are added on the queue id received as param
     - filter all the competence levels from the above, meaning comparing the filters received with the filters added on each competence level. In case some competence levels have no filters set (even if we receive filters param) those competence levels are automatically added on list of result.

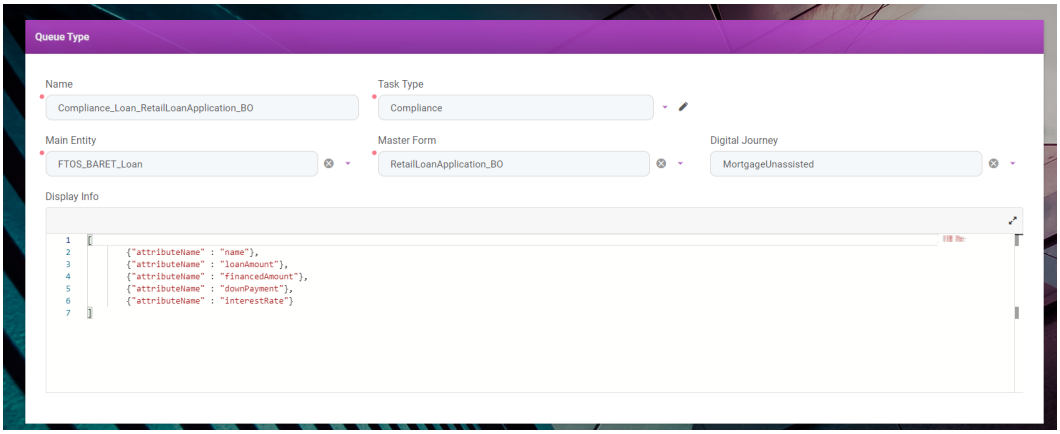


5. `attachToQueue(queueItemId, queueTypeId, taskTypeId, JSON_filterQueue, JSON_filterCompetence)`
   - ***Input***:
     - `queueItemId` - the id of the queue item which will be added to a queue
     - `queueTypeId` - the queue type of the item which will be added to a queue
     - `taskTypeId` - the task type of the item which will be added to a queue
     - `JSON_filterQueue` - the filter on which the queues will be filtered
     - `JSON_filterCompetence` - the filter on which the competence Levels will be filtered and the required competence level id will be decided (the smallest from the list)
   - ***Output***: no result as output, but the outcome of this function is that the queue item will be added to a queue, which means that the followings will be updated:
     - queueId
     - description
     - requiredCompetenceLevelId
   - ***CallExample***:

```
var queueFilter= [];
var productTypeFilter = {};
productTypeFilter.name = "ProductType";
productTypeFilter.value = "MG";
queueFilter.push(productTypeFilter);

var competenceLevelFilter= [];
var requestedLoanAmountFilter = {};
requestedLoanAmountFilter .name = "RequestedLoanAmount";
requestedLoanAmountFilter .value = "57000";
competenceLevelFilter.push(requestedLoanAmountFilter );

var queueTypeId = 'e398ff08-9127-482d-b126-3758bb42b08d';
var taskTypeId = 'a198ff08-9127-482d-b126-3758bb42b08d';

attachToQueue(queueItemId, queueTypeId, null, queueFilter,
competenceLevelFilter);
//or with taskTypeId
attachToQueue(queueItemId, null, taskTypeId, queueFilter,
competenceLevelFilter);
```

   - ***Technical description:***
     - based on queueItemId, it will be retrieved recordId and uniqueId which will be later on used in implementation
     - call function `getAvailableQueue(JSON_filterQueue, queueTypeId, taskTypeId)` to get the queue which this queue item will be added on

- using attribute ‘displayFields' from queue, format the 'description’ attribute for queueItem. An example for displayFields is:



  - Using that attribute, it will be generated a description from the main entity (also set on QueueType entity)
  - The final description will be like this:

- call function `getCompetenceLevels(JSON_filterCompetence, queue.queueId)` to get the list of competence levels for the current queue and available operators; order the list of competence levels by level number ascending and the first one will be the required competence level for this queue item
- update queueItem with queueId, description and requiredCompetenceLevelId

6. `attachOperatorToQueueItem(queueItem, profileFilter, competenceLevelId)`
   - ***Input***:
     - `queueItem` - queue item which will be attached to an operator. Mandatory attributes for this object:
       - `FTOS_CMB_QueueItemid`
       - `QueueId`
       - `UniqueId`
       - `RecordId`
       - `RecordDate`
       - `RecordName`
       - `ReturnToSameOperator`
       - `BusinessStatusId`
     - `profileFilter` - on which the profiles of operators will be filtered
     - `competenceLevelId` - the competence level necessary for attaching to an operator
   - ***Output***: no result as output, but the outcome of this function is that an operator will be assigned to this queue item using function `assignOperatorToQueueItem`
   - ***CallExample***:

```
var filtersParams = [];
var customerSegmentFilter = {};
customerSegmentFilter.name = "Customer Segment";
customerSegmentFilter.value = "Default";
filtersParams.push(productTypeFilter);
var skillFilter = {};
skillFilter.name = "Skill";
skillFilter.value = "eb44ba02-7428-4c60-9fa6-7e850435205b";
filtersParams.push(skillFilter);
//filters that are lookup, the value needs to be the ids;
//for option sets the value will be the name of the option set
item

var competenceLevelId = 'eb44ba02-7428-4c60-9fa6-7e850435205b';

var queueItem={};
queueItem.FTOS_CMB_QueueItemid = "eb44ba02-7428-4c60-9fa6-
7e850435205b";
queueItem.QueueId = 'db44ba02-7428-4c60-9fa6-7e850435205b';
queueItem.UniqueId = '2920202111111';
queueItem.RecordId = 'bb44ba02-7428-4c60-9fa6-7e850435205b';
queueItem.RecordDate = '';
queueItem.RecordName = 'Application 0001004';
queueItem.ReturnToSameOperator = false;
queueItem.BusinessStatusId = 'ab44ba02-7428-4c60-9fa6-
7e850435205b'

attachOperatorToQueueItem(queueItem,filtersParams,
competenceLevelId);
```

- ***Technical description:***
  - get replacementCompetenceLevelId if exists - on CompetenceLevel entity
  - filter profiles and get one profile
  - trying to get an operator
    - if the queue item is in review - trying to get an operator with the specific competence level id passed through parameters. If no operator found, the allocation will be not be done, because in review is mandatory to be reviewed by that specific competence level
    - if queue item not in review, the allocation will be tried in the following order:
      - try to find a queue item or queue item processed with the same record id and the operator attach to it is ready. Sort descending the resulted list by created date of the queue item; if any, call function `assignOperatorToQueueItem` to assign operator to this queueitem
      - if ReturnToSameOperator attribute from queueItem param (it's actually a related attribute from queue) is true, try to find a queue item or queue item processed with the same unique id, with an operator ready and in the last x months(system paramter LastXMonths). Sort descending the resulted list by created date of the queue item; if any, call function `assignOperatorToQueueItem` to assign operator to this queueitem
      - try to get an available operator using function `getAvailableOperator` with the competenceLevelId received from param.
        - if found, call function `assignOperatorToQueueItem` to assign operator to this queueitem

- If no one found and this competence level has a replacement competence level, try to find an operator using the replacementCompetenceLevelId given as param to function `getAvailableOperator`. If found, call function `assignOperatorToQueueItem` to assign operator to this queueitem
- if any of the above tactics of finding an available operator fails, then the allocation will not be done

7. `assignOperatorToQueueItem(operatorId, queueItemId, queueId, operatorProfileId, operatorCompetenceLevelId, isReturned)`
   - ***Input***:
     - `operatorId` - operator id on which the queue item will be allocated
     - `queueItemId` - the id of the queue item which will be allocated
     - `queueId` - queue id on which the queue item is already allocated
     - `operatorProfileId` - the operator Profile id matching the current queue item - not mandatory
     - `operatorCompetenceLevelId` - the competence level id of the operator which matches the current queue item - not mandatory
     - `isReturned` - a flag which specify if the queue item is returned or not - not mandatory
   - ***Output***: no result as output, but the outcome of this function is that queue item will be updated, the status of it will be changed to allocated and also some details will be updated on table FTOS_CMB_OperatorXQueue
   - ***CallExample***:

```
var operatorId = 'ab44ba02-7428-4c60-9fa6-7e850435205b';
var queueItemId = 'bb44ba02-7428-4c60-9fa6-7e850435205b';
var queueId= 'cc44ba02-7428-4c60-9fa6-7e850435205b';
var operatorProfileId= 'cc44ba02-7428-4c60-9fa6-
7e850435205b';
var operatorCompetenceLevelId= 'dd44ba02-7428-4c60-9fa6-
7e850435205b';
var isReturned = false;

assignOperatorToQueueItem(operatorId, queueItemId, queueId,
operatorProfileId, operatorCompetenceLevelId,isReturned);
//or without not mandatory params
assignOperatorToQueueItem(operatorId, queueItemId, queueId);
```

   - ***Technical description:***
     - update `FTOS_CMB_QueueItem` with operator id, allocation date(as current date), profile id(if received as param), operator competence level id (if received as param), isReturn (set to true if received as param)
     - update `FTOS_CMB_OperatorXQueue` by increasing the item count; last allocated date will be set as current date only if the business status of the queue item is not ‘Review' - basically do this only for 'New’ statuses
     - change business status of the queue item to “Allocated“
8. `getMainEntityByQueueType(queueTypeId)`
   - ***Input***: `queueTypeId` - queueTypeId on which the priority is set
   - ***Output***: `entityName` - main entity set on Queue level
   - ***CallExample***: `var res = getMainEntityByQueueType(context.Data.queueTypeId);`
   - ***Technical description:*** Get the main entity name
9. `getMainEntityByQueue(queueId)`
   - ***Input***: `queueId` - queueId on which the priority is set
   - ***Output***: `entityName` - main entity set on Queue level
   - ***CallExample***: `res = getMainEntityByQueue(context.Data.queueId);`
   - ***Technical description:*** Get the main entity name
10. `dynamicJoinAndOrderBy(operatorId, aliasQueueItem)`
    - ***Input***:
      `operatorId`- operator id on which the queue item is allocated
      `aliasQueueItem`-FTOS_CMB_QueueItem alias
11. ***Output***:

a. ***CallExample***:
`var result = dynamicJoinAndOrderBy('88e5634-22bf-4e77-9e23-ebdba5996fb5', 'Q');`
Output example


```
{
   "orderBySequence":[
      {
         "type":"asc",
         "attribute":"OQ.queuePriority"
      },
      {
         "type":"desc",
         "attribute":"FTOS_BARET_Loan.createdOn"
      },
      {
         "type":"desc",
         "attribute":"Q.allocationDate"
      }
   ],
   "joinSequence":[
      {
         "type":"left",
         "entity":{
            "alias":"FTOS_BARET_Loan",
            "name":"FTOS_BARET_Loan",
            "attributelist":[

            ]
         },
         "fromto":[
            {
               "from":"recordId",
               "to":"FTOS_BARET_Loanid"
            }
         ]
      }
   ]
}
```


b. ***Technical description:***
Generates the join and where section with the queue item entity(Ex: FTOS_BARET_Loan)

12. `getOperatorCompetenceLevel(operatorId)`
    - ***Input***:
      `operatorId`- operator id
    - ***Output***:
    - ***CallExample***:
      `var result = getOperatorCompetenceLevel('88e5634-22bf-4e77-9e23-ebdba5996fb5');`
    - ***Technical description:***
      Get the operator competence level.
13. `getFlowSettings(digitalJourneyId, digitalProcessorTypeName)`
    - ***Input***:
      - `digitalJourneyId` - digital journey id on which the flow setting is set

- `digitalProcessorTypeName` - processor type name on which the filter settings are
- ***Output***: an object with the followings properties:
  - `ProcessorSettingsId` - the id `FTOS_DFP_ProcessorSettingsid`
  - `settings` - from processorSettings attribute settings
  - `mapping` - from processorSettings attribute settings
- ***CallExample***:

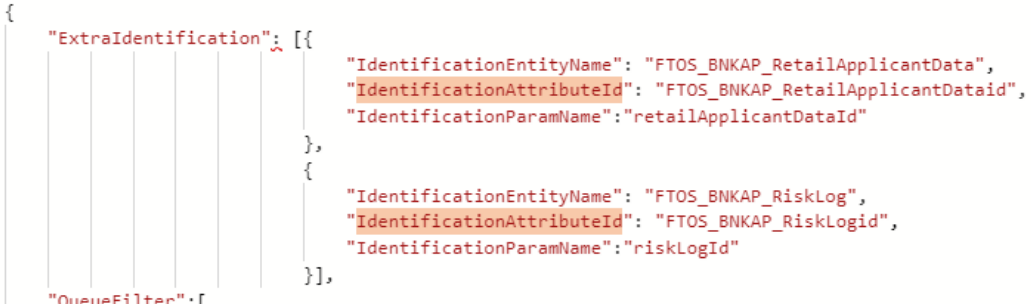
```
var result = getFlowSettings('ab44ba02-7428-4c60-9fa6-
7e850435205b', 'ComplianceFilters');
```

- ***Technical description:***
  - get the flow settings for specific digital journey id and digital processor type name, selecting attributes settings and maps

14. `getFiltersUsingFlowSetttings(queueItemRecordId, queueTypeId, extraIdentifications, digitalProcessorTypeName)`
    - ***Input***:
      - `queueItemRecordId` - record id from queue item
      - `queueTypeId` - queue type id
      - `extraIdentifications` - extra identifications received as params will be used in queries for getting the filters
      - `digitalProcessorTypeName` - digital processor type name - will be used in calling function `getFlowSettings`
    - ***Output***: an object with the followings properties:
      - `queueFilter`
      - `competenceFilter`
      - `profileFilter`
    - ***CallExample***:

```
var queueItemRecordId = 'ab44ba02-7428-4c60-9fa6-
7e850435205b';
var queueTypeId= 'bb44ba02-7428-4c60-9fa6-7e850435205b';
var digitalProcessorTypeName= 'ComplianceFilters';
var extraIdentifications= [
  {retailApplicantDataId:  'cc44ba02-7428-4c60-9fa6-
7e850435205b',
   riskLogId: 'dd44ba02-7428-4c60-9fa6-7e850435205b'}];


var filters = getFiltersUsingFlowSetttings(queueItemRecordId,
queueTypeId, extraIdentifications, digitalProcessorTypeName);
```

    - ***Technical description:***
      - function `getFlowSettings` is called for getting the flow settings. The flow settings for Filters should look something like



```
    {
        "filterName":"ProductType",
        "isOptionSet":true,
        "optionSetName":"bpProductType",
        "entityName":"FTOS_BARET_Loan",
        "attributeName":"productTypeId"
    }
  ]
,
 "ProfileFilter":[
    {
        "filterName":"Customer Segment",
        "isOptionSet":true,
        "optionSetName":"FTOS_ACC_CustomerSegment",
        "entityName":"FTOS_BNKAP_RetailApplicantData",
        "attributeName":"customerSegmentId"
    },
    {
        "filterName":"Skill",
        "isOptionSet":false,
        "entityName":"FTOS_BNKAP_RiskLog",
        "attributeName":"riskListId"
    }
  ]
,
 "CompetenceLevelFilter":[
    {
        "filterName":"RequestedLoanAmount",
        "isOptionSet":false,
        "entityName":"FTOS_BARET_Loan",
        "attributeName":"loanAmount"
    }
  ]
}
```


- for each filter meaning, for each object: ‘QueueFilter', ‘ProfileFilter’ and 'CompetenceLevelFilter’ are called the following functions:
  - `checkIfRelatedEntityOrOptionSetIsOk`
  - `getValueFromRelatedEntityOrOptionSet`
- the main idea of this function is to generally retrieve the values for each filter and giving the possibility to define at flow settings the filters

15. `checkIfRelatedEntityOrOptionSetIsOk(filter, mainEntityTableName)`
    - ***Input***:
      - `filter` - object which contains properties: filterName, entityName, isOptionSet, optionSetName, attributeName
      - `mainEntityTableName` - table name of the main entity set on QueueType (e.g. 'FTOS_BARET_Loan')
    - ***Output***: an object containing the following properties:
      - `isOk` - if true, the entity name, option set name and attribute name exists (basically correctly added on flow settings)
      - `entityName` - entity name on which the fetch needs to be done to get the value
      - `refAttributeName` - the attribute name on which the link is made between main entity and current entity
    - ***CallExample***:

```
var mainEntityTableName = 'FTOS_BARET_Loan';
var filter ={filterName: 'Skill', isOptionSet: false,
entityName: 'FTOS_BNKAP_RiskLog', attributeName: 'riskListId'};

var result= checkIfRelatedEntityOrOptionSetIsOk(filter,
mainEntityTableName);
```

    - ***Technical description:***

- in case the entity name is the same as the entityName return `{ isOk: true, entityName: filter.entityName, refAttributeName: filter.entityName + 'id' };`
- check if entity name from filter is related with main entity using db task `FTOS_IsRelatedEntities`. If yes, result will be`{ isOk: true, entityName: rows.Records[0].referencedE, refAttributeName: rows.Records[0].referencedA }`
- if it's option set, check if that option set name exists
- at the end check if the attribute name from filter exists in the entity name of the filter

16. `getValueFromRelatedEntityOrOptionSet(filter, mainEntityTableName, referencedEandA, recordId, filterExtraIdentificationList, extraIdentifications)`
    - ***Input***:
        - `filter` - object which contains properties: filterName, entityName, isOptionSet, optionSetName, attributeName
        - `mainEntityTableName` - the table name of the main entity set on queue type
        - `referencedEandA` - an object with contains properties: `entityName` and `refAttributeName`
        - `recordId` - the record id of the task
        - `filterExtraIdentificationList` - extra identification list from flow settings containing properties: `IdentificationAttributeId`, `IdentificationEntityName` and `IdentificationParamName`
        - `extraIdentifications` - extra identifications values
    - ***Output***:
        - the value for a specific filter
    - ***CallExample***:

```
var filter ={ filterName: 'Skill',
              isOptionSet: false,
              entityName: 'FTOS_BNKAP_RiskLog',
              attributeName: 'riskListId'};
var mainEntityTableName = 'FTOS_BARET_Loan';
var referencedEandA = { isOk: true,
                        entityName: "FTOS_BARET_Loan",
                        refAttributeName: "FTOS_BARET_Loanid"};
var recordId = 'bf29264f-eb0b-4dcf-9eb9-0c9b7f48a2e3';
var filterExtraIdentificationList = [
  {
    IdentificationEntityName: "FTOS_BNKAP_RetailApplicantData",
    IdentificationAttributeId:
"FTOS_BNKAP_RetailApplicantDataid",
    IdentificationParamName: "retailApplicantDataId"
  },
  {
    IdentificationEntityName: "FTOS_BNKAP_RiskLog",
    IdentificationAttributeId: "FTOS_BNKAP_RiskLogid",
    IdentificationParamName: "riskLogId"
  }
];
var extraIdentifications  = {
  retailApplicantDataId: "631def4a-c0ab-4c3f-9a23-
f5ffc4488f13",
  riskLogId: "2437faec-3f44-437d-a7d7-fcbdd0907d15"
};

var value = getValueFromRelatedEntityOrOptionSet(filter ,
mainEntityTableName, referencedEandA, recordId,
filterExtraIdentificationList, extraIdentifications);
```

- ***Technical description:***
  - by default the where condition will be using the attribute name from `referencedEandA.refAttributeName` and the value will be the record id
  - depending on the filterExtraIdentificationList it will be decided if the 'where' condition will be the one from extra identification field.
    - the check is done by verifying if in list `filterExtraIdentificationList` there is one element with the IdentificationEntityName equals to referencedEandA.entityName
    - in case the above result to using th extra identification the where attribute will be `filterExtraIdentification.IdentificationAttributeId` and the value will be retrieved from `extraIdentifications` using `filterExtraIdentification.IdentificationParamName`
  - if it's option set, a general fetch will be executed by joining the `referencedEandA.entityName` with the optionsetitem, conditioned by the where decided as explained above and selecting the 'name' attribute from optionsetitem
  - if not an option set, a general fetch will be executed from`referencedEandA.entityName`, conditioned by the where decided as explained above and selecting the attribute `'filter.attributeName'`

17. `createAndAllocateQueueItem(extraIdentifications, queueTypeId, recordId, uniqueId, blnIsReview, digitalProcessorTypeName)`
    - ***Input***:
      - `extraIdentifications` - used in function `getValueFromRelatedEntityOrOptionSet` for getting the values for filters
      - `queueTypeId` - queue type id which matches the current record added as task
      - `recordId` - record id of the task which will be added
      - `uniqueId` - unique id of the task which will be added
      - `blnIsReview` - if this set to true, it means this will be a second task added for the same record only for review reasons
      - `digitalProcessorTypeName`- digital processor type name used for getting the flow settings used for filters
      - `operatorId` - operator id, it's not mandatory and if given will override the logic of finding an operator and automatically allocate the task to that operator
      - `operatorProfileId` - operator profile id, it's not mandatory; filled in as operator profile id on queue item in the case of manually allocate the task to a specific operator
      - `dueDays` - due days, used for calculate the due date (deadline) on queue item; if given will override the due days from queue type
    - ***Output***: no result as output, but the outcome of this function is:
      - a queue item is created
      - the queue item is added to a queue
      - the queue item will be allocated to an operator
    - ***CallExample***:

```
var extraIdentifications  = {
  retailApplicantDataId: "631def4a-c0ab-4c3f-9a23-
f5ffc4488f13",
  riskLogId: "2437faec-3f44-437d-a7d7-fcbdd0907d15"
};
var queueTypeId = 'e398ff08-9127-482d-b126-3758bb42b08d';
var recordId= '64f1c182-d329-4d94-892e-c3cf4556d186';
var uniqueId= '1313131313';
var blnIsReview = false;
var digitalProcessorTypeName = 'ComplianceFilters';

createAndAllocateQueueItem(extraIdentifications, queueTypeId,
recordId, uniqueId, blnIsReview, digitalProcessorTypeName);
```

    - ***Technical description:***
      - first validate parameters
      - get queueType info: main entity id, main form id, digital journey id
      - get record name and record date using record id and main entity

- create queue item setting the following attributes:
  - `mainEntityId`
  - `mainFormId`
  - `recordId`
  - `uniqueId`
  - `recordName`
  - `recordDate`
  - `businessStatusId` - set to 'Review' only when param `blnIsReview` is set to 'true', otherwise do not set this attribute so the default one be assigned, in this case 'New'
  - `previousQueueItemProcessedId` or `previousQueueItemId` - will be set with a value retrieved from calling function `getPrevousQueueItemId(recordId)`
- get filters by calling function `getFiltersUsingFlowSetttings(recordId, queueTypeId, extraIdentifications, digitalProcessorTypeName)`
- try to add the queue item to a queue by calling function `attachToQueue(insertedQueueItemId, queueTypeId, null, filters.queueFilter, filters.competenceFilter)`
- get the first competence level which needs to be used by calling function `getCompetenceLevels(filters.competenceFilter, queueItem.QueueId)` and then sort them ascending by level
- depending on received parameter `operatorId`
  - if exists, the queue item will be assigned to that specific operator Id
  - if not, find an operator try to attach the queue item to it by calling function `attachOperatorToQueueItem (queueItem, filters.profileFilter, competenceLevels[0].itemId);`

18. `getPrevousQueueItemId(recordId)`
    - ***Input***:
      - `recordId`
    - ***Output***: an object which represents the previous queue item with the following attributes:
      - `PreviousItemId`
      - `isProcessed`
    - ***CallExample***:

```
var recordId= '64f1c182-d329-4d94-892e-c3cf4556d186';

var previousQueueItem = getPrevousQueueItemId(recordId);
```

    - ***Technical description:***
      - try to find in both entities `FTOS_CMB_QueueItem` and `FTOS_CMB_QueueItemProcessed` an item with the same record id; order them descending by `CreatedOn`
      - if any found in `FTOS_CMB_QueueItem`, return an object with properties:
        - `PreviousItemId` - the id of the `FTOS_CMB_QueueItem`
        - `isProcessed` - false
      - else if found in `FTOS_CMB_QueueItemProcessed`, return an object with properties:
        - `PreviousItemId`- the id of the `FTOS_CMB_QueueItemProcessed`
        - `isProcessed` - true
19. `validateDisplayInfo(mainEntityId, displayInfo)`
    - ***Input***: `mainEntityId` - main entity set on QueueType level, `displayInfo` - displayFields set on QueueType level
    - ***Output***:
      - If everything is ok, the result will be an object with `isSuccess = true;` and `message = 'Ok';`
      - If there are attributes in display info that are not contain in the main entity, then the result will be `isSuccess = false;` and `message = One or more attributes are not included in master entity!`
      - If the json format is not correct, then the result will be `isSuccess = false;` and `message = Display info format is incorrect!`
    - ***CallExample***:

```
var mainEntityId = context.Values.mainEntityId;
var displayFields = context.Values.displayFields;
```

```
var resultValidation = validateDisplayInfo(mainEntityId,
displayFields);
```

- ***Technical description:***
  - This function will validate the list of attributes given in displayInfo parameter, in JSON format, against the list of attributes of the main entity, given as a parameter.

20. `sortValues(key, order)`
    - ***Input***:
      - key - key on which the comparisson value will be used
      - order - ‘asc’ or ‘desc’
    - ***Output***:
      - comparison number
    - ***CallExample***:

```
var competenceLevels= [
  {
    itemId: "631def4a-c0ab-4c3f-9a23-f5ffc4488f13",
    itemName: "Competence Level 3",
    level: 3
  },
  {
    itemId: "2437faec-3f44-437d-a7d7-fcbdd0907d15",
    itemName: "Competence Level 3",
    level: 1
  },
  {
    itemId: "64f1c182-d329-4d94-892e-c3cf4556d186",
    itemName: "Competence Level 2",
    level: 2
  }
];

competenceLevels.sort(sortValues("level", "asc"));
```

    - ***Technical description:***
      - comparison function for sorting values in an array of objects on a specific key. This will be used as the param in the general .sort function

21. `contains(arr, key, val)`
    - ***Input***:
      - `arr` - an array of objects
      - `key` - the param name of the object
      - `val` - the value on which the condition is made
    - ***Output***:
      - if the array contains an object with that specific value for that key, returns the object, otherwise return null
    - ***CallExample***:

```
var filterExtraIdentificationList = [
  {
    IdentificationEntityName: "FTOS_BNKAP_RetailApplicantData",
    IdentificationAttributeId:
"FTOS_BNKAP_RetailApplicantDataid",
```

```
    IdentificationParamName: "retailApplicantDataId"
  },
  {
    IdentificationEntityName: "FTOS_BNKAP_RiskLog",
    IdentificationAttributeId: "FTOS_BNKAP_RiskLogid",
    IdentificationParamName: "riskLogId"
  }
];

var referencedEandA = { isOk: true,
                        entityName:
"FTOS_BNKAP_RetailApplicantData",
                        refAttributeName:
"FTOS_BNKAP_RetailApplicantDataid"};

var filterExtraIdentification = contains
(filterExtraIdentificationList, "IdentificationEntityName",
referencedEandA.entityName);
//filterExtraIdentification will be
//{
//    IdentificationEntityName:
"FTOS_BNKAP_RetailApplicantData",
//    IdentificationAttributeId:
"FTOS_BNKAP_RetailApplicantDataid",
//    IdentificationParamName: "retailApplicantDataId"
//  }
```

- ***Technical description:***
  - iterate through array and if the condition is satisfied, return the object; otherwise return null.

22. `operatorIsOnQueue(operatorId, queueId)`
    - ***Input***:
      - `operatorId`- operator id which is checked if exists on a specific queue id
      - `queueId`- queue id on which the operator is checked
    - ***Output***:
      - true if operator is assigned on that queue id; otherwise, false
    - ***CallExample***:

```
operatorIsOnQueue('64f1c182-d329-4d94-892e-c3cf4556d186',
'2437faec-3f44-437d-a7d7-fcbdd0907d15');
```

    - ***Technical description:***
      - a fluent query is made for checking if the operator id is assigned on a specific queue
23. `getHighestCompetenceLevelOfOperator(operatorId, queueId)`
    - ***Input***:
      - `operatorId`- operator id from which the highest competence level is retrieved
      - `queueId`-queue id from which the highest competence level is retrieved
    - ***Output***:
      - the id of the highest competence level is found any, null otherwise
    - ***CallExample***:

```
```

```
getHighestCompetenceLevelOfOperator('64f1c182-d329-4d94-892e-
c3cf4556d186', '2437faec-3f44-437d-a7d7-fcbdd0907d15');
```

- ***Technical description:***
  - all competence levels from a specific operator and queue are retrieved, sort them descending by level and return the first one found.