

InterestRate_CapFloor

Complete Product Qualification Flow for InterestRate_CapFloor

1. Overview and Context

Product qualification in CDM is inferred from economic terms, not explicit product type declarations. The system uses qualification functions annotated with `[qualification Product]` to determine product types based on ISDA Taxonomy v2.0.

For `InterestRate_CapFloor`, the qualification identifies:

- Interest Rate Cap (only cap rate schedule)
- Interest Rate Floor (only floor rate schedule)
- Interest Rate Collar (both cap and floor rate schedules)

2. Qualification Function Structure

The main qualification function is:

Location: `common-domain-model-5.27.0/rosetta-source/src/main/rosetta/product-qualification-func.rosetta`
(Lines 1447-1461)

```
func Qualify_InterestRate_CapFloor:
  [qualification Product]
  inputs:
    economicTerms EconomicTerms (1..1)
  output:
    is_product boolean (1..1)
    [synonym ISDA_Taxonomy_v1 value "InterestRate_CapFloor"]
    [synonym ISDA_Taxonomy_v2 value "InterestRate_CapFloor"]

  set is_product:
    Qualify_AssetClass_InterestRate(economicTerms) = True
```

and economicTerms → payout → interestRatePayout count = 1
and (economicTerms → payout → interestRatePayout → rateSpecification
on → floatingRate → capRateSchedule exists
or economicTerms → payout → interestRatePayout → rateSpecification
on → floatingRate → floorRateSchedule exists)

3. Step-by-Step Qualification Flow

Step 1: Asset Class Qualification (Prerequisite)

First, `Qualify_AssetClass_InterestRate` must return `True`.

Location: Lines 24-54 of `product-qualification-func.rosetta`

Logic:

- Primary path: `economicTerms → payout → interestRatePayout only exists`
- Alternative paths:
 - Option-based: If `optionPayout` exists, the underlier must be:
 - A Debt security, OR
 - A security/index with `primaryAssetClass = InterestRate`, OR
 - Recursively qualify as Interest Rate through its economic terms
 - Forward-based: If `forwardPayout` exists (with or without `interestRatePayout / cashflow`), the underlier must be:
 - A Debt security, OR
 - A security/index with `primaryAssetClass = InterestRate`

For CapFloor: Typically satisfied by having `interestRatePayout` present.

Step 2: Single Interest Rate Payout Check

`economicTerms → payout → interestRatePayout count = 1`

- Must have exactly one `interestRatePayout` (not zero, not multiple).
- This distinguishes CapFloor from swaps (which typically have 2+ legs).

Step 3: Cap/Floor Schedule Existence Check

economicTerms → payout → interestRatePayout → rateSpecification → floatingRate → capRateSchedule exists

or

economicTerms → payout → interestRatePayout → rateSpecification → floatingRate → floorRateSchedule exists

Data Path Navigation:

1. economicTerms → payout → interestRatePayout (single element)
2. interestRatePayout → rateSpecification (must exist)
3. rateSpecification → floatingRate (must exist; not fixedRate or inflationRate)
4. floatingRate → capRateSchedule OR floorRateSchedule (at least one must exist)

Structure Details:

RateSpecification (Lines 223-230 of product-asset-type.rosetta):

- One of: fixedRate, floatingRate, or inflationRate
- For CapFloor: must be floatingRate

FloatingRate (extends FloatingRateBase, Lines 702-706):

- Inherits from FloatingRateBase (Lines 693-700)
- capRateSchedule : StrikeSchedule (0..1) — optional cap rate or schedule
- floorRateSchedule : StrikeSchedule (0..1) — optional floor rate or schedule

StrikeSchedule:

- Contains a price (or price schedule) with strike values
- Includes buyer and seller roles
- Can be a single rate or a schedule with dates

4. Data Model Structure

Complete Path:

EconomicTerms

- └─ payout: Payout
 - └─ interestRatePayout: InterestRatePayout[1] (exactly one)
 - └─ rateSpecification: RateSpecification
 - └─ floatingRate: FloatingRateSpecification
 - └─ capRateSchedule: StrikeSchedule (0..1) [REQUIRED FOR CAP/COLLAR]
 - └─ floorRateSchedule: StrikeSchedule (0..1) [REQUIRED FOR FLOOR/COLLAR]

5. Qualification Scenarios

Scenario A: Interest Rate Cap

- `Qualify_AssetClass_InterestRate` = True
- `interestRatePayout count` = 1
- `capRateSchedule exists` = True
- `floorRateSchedule exists` = False
- Result: Qualified as `InterestRate_CapFloor`

Scenario B: Interest Rate Floor

- `Qualify_AssetClass_InterestRate` = True
- `interestRatePayout count` = 1
- `capRateSchedule exists` = False
- `floorRateSchedule exists` = True
- Result: Qualified as `InterestRate_CapFloor`

Scenario C: Interest Rate Collar

- `Qualify_AssetClass_InterestRate` = True
- `interestRatePayout count` = 1
- `capRateSchedule exists` = True

- `floorRateSchedule exists` = True
- Result: Qualified as `InterestRate_CapFloor`

6. Example Data Structure

From `ird-ex22-cap.json` (Cap example):

```
{
  "rateSpecification": {
    "floatingRate": {
      "rateOption": { ... },
      "capRateSchedule": {
        "price": { "address": { "scope": "DOCUMENT", "value": "price-1" } },
        "buyer": "Receiver",
        "seller": "Payer"
      }
    }
  }
}
```

From `ird-ex24-collar.json` (Collar example):

```
{
  "rateSpecification": {
    "floatingRate": {
      "rateOption": { ... },
      "capRateSchedule": { ... },
      "floorRateSchedule": { ... }
    }
  }
}
```

7. Execution Context

How Qualification Functions Are Invoked:

1. All qualification functions with `[qualification Product]` are collected for `EconomicTerms`.

2. Each function is evaluated against the `EconomicTerms` instance.
3. Functions returning `True` indicate a match.
4. Multiple qualifications are possible (e.g., a Swaption can qualify as both an Option and the underlying Swap).
5. The result is typically stored in `productTaxonomy.productQualifier`.

Implementation Pattern (from `Qualification.java`):

```
// Extract qualification functions
var qualifyFunctions = new EconomicTermsMeta().getQualifyFunctions(qualify
FunctionFactory);

// Evaluate and extract results
var qualificationResult = new QualifyResultsExtractor<>(qualifyFunctions, eco
nomicTerms)
    .getOnlySuccessResult()
    .map(QualifyResult::getName)
    .orElse("Failed to qualify");
```

8. Key Validation Points

1. Must be an Interest Rate product (via `Qualify_AssetClass_InterestRate`).
2. Must have exactly one `interestRatePayout`.
3. Must have a `rateSpecification` with a `floatingRate` (not fixed or inflation).
4. Must have at least one of `capRateSchedule` or `floorRateSchedule` present.
5. The `floatingRate` must exist (cannot be null/absent).

9. Taxonomy Mapping

- ISDA Taxonomy v1.0: `InterestRate_CapFloor`
- ISDA Taxonomy v2.0: `InterestRate_CapFloor`
- Both are represented as synonyms in the function output.

10. Important Notes

1. The function does not distinguish between Cap, Floor, and Collar; all three qualify as `InterestRate_CapFloor`.
2. The qualification is based on structure, not explicit product type.
3. A product can qualify as multiple types if it matches multiple functions.
4. The qualification is deterministic based on economic terms.
5. The `StrikeSchedule` can be a single rate or a schedule with multiple dates/rates.

This flow ensures that any product with the required structure is correctly identified as an Interest Rate CapFloor product according to ISDA taxonomy standards.

COMPLETE STEP-BY-STEP PRODUCT QUALIFICATION FLOW FOR `InterestRate_CapFloor`

PART 1: FUNCTION DEFINITION AND STRUCTURE

1.1 The Qualification Function Location

File: `common-domain-model-5.27.0/rosetta-source/src/main/rosetta/product-qualification-func.rosetta`

Lines: 1447-1461

Code Evidence:

```
1447|func Qualify_InterestRate_CapFloor: <"Qualifies a product as an interest
    |rate cap, interest rate floor, or an interest rate collar based on the economic te
    |rms and the following criteria: 1) An interest rate product with one one leg that
    |includes a cap and/or a floor.">
1448|  [qualification Product]
1449|  inputs:
1450|    economicTerms EconomicTerms (1..1)
1451|  output:
1452|    is_product boolean (1..1)
1453|      [synonym ISDA_Taxonomy_v1 value "InterestRate_CapFloor"]
1454|      [synonym ISDA_Taxonomy_v2 value "InterestRate_CapFloor"]
1455|
```

```

1456| set is_product:
1457|     Qualify_AssetClass_InterestRate(economicTerms) = True
1458|         and economicTerms → payout → interestRatePayout count = 1
1459|         // qualifies the product as having a cap and/or floor in the interestRatePayout
1460|         and economicTerms → payout → interestRatePayout → rateSpecification → floatingRate → capRateSchedule exists
1461|         or economicTerms → payout → interestRatePayout → rateSpecification → floatingRate → floorRateSchedule exists

```

Explanation:

- Line 1447: Function name follows `Qualify_` + taxonomy name pattern.
- Line 1448: `[qualification Product]` marks this as a product qualification function.
- Line 1450: Input is a single `EconomicTerms` instance (required).
- Line 1452: Output is a boolean indicating qualification.
- Lines 1453-1454: Synonyms map to ISDA Taxonomy v1 and v2.
- Lines 1456-1461: The qualification logic.

PART 2: DATA MODEL STRUCTURE - EXACT TYPE DEFINITIONS

2.1 EconomicTerms Type Definition

File: `common-domain-model-5.27.0/rosetta-source/src/main/rosetta/product-template-type.rosetta`

Lines: 32-50

Code Evidence:

```

32|type EconomicTerms: <" This class represents the full set of price-forming
features associated with a contractual product: the payout component, the notional/quantity, the effective and termination date and the date adjustment provisions when applying uniformly across the payout components. This class also includes the legal provisions which have valuation implications: cancelable

```

```

provision, extendible provision, early termination provision and extraordinary
events specification.">
33|
34|   effectiveDate AdjustableOrRelativeDate (0..1)
35|   terminationDate AdjustableOrRelativeDate (0..1)
36|   dateAdjustments BusinessDayAdjustments (0..1)
45|   payout Payout (1..1) <"The payout specifies the future cashflow computa
tion methodology which characterizes a financial product.">
46|   terminationProvision TerminationProvision (0..1)
47|   calculationAgent CalculationAgent (0..1)
48|   nonStandardisedTerms boolean (0..1)
50|   collateral Collateral (0..1)

```

Explanation:

- Line 45: `payout Payout (1..1)` means `payout` is required and exactly one.

2.2 Payout Type Definition

File: `common-domain-model-5.27.0/rosetta-source/src/main/rosetta/product-template-type.rosetta`

Lines: 289-302

Code Evidence:

```

289|type Payout: <"A class to represent the set of future cashflow methodolo
gies in the form of specific payout class(es) that can be associated for the pur
pose of specifying a financial product. For example, two interest rate payouts
can be combined to specify an interest rate swap, or one interest rate payout
can be combined with a credit default payout to specify a credit default swa
p.">
290|   [metadata key]
291|
292|   interestRatePayout InterestRatePayout (0..*) <"All of the terms necessa
ry to define and calculate a cash flow based on a fixed, a floating or an inflatio
n index rate. The interest rate payout can be applied to interest rate swaps an
d FRA (which both have two associated interest rate payouts), credit default s

```

waps (to represent the fee leg when subject to periodic payments) and equity swaps (to represent the funding leg).">

Explanation:

- Line 292: `interestRatePayout InterestRatePayout (0..*)` means zero or more elements (collection).

2.3 InterestRatePayout Type Definition

File: `common-domain-model-5.27.0/rosetta-source/src/main/rosetta/product-asset-type.rosetta`

Lines: 135-153

Code Evidence:

```
135|type InterestRatePayout extends PayoutBase: <" A class to specify all of t  
he terms necessary to define and calculate a cash flow based on a fixed, a flo  
ating or an inflation index rate. The interest rate payout can be applied to inter  
est rate swaps and FRA (which both have two associated interest rate payout  
s), credit default swaps (to represent the fee leg when subject to periodic pay  
ments) and equity swaps (to represent the funding leg). The associated global  
Key denotes the ability to associate a hash value to the InterestRatePayout ins  
tantiations for the purpose of model cross-referencing, in support of functiona  
lity such as the event effect and the lineage.">
```

```
136|  [metadata key]
```

```
137|
```

```
138|  rateSpecification RateSpecification (0..1) <"The specification of the rate  
value(s) applicable to the contract using either a floating rate calculation, a sin  
gle fixed rate, a fixed rate schedule, or an inflation rate calculation.">
```

Explanation:

- Line 138: `rateSpecification RateSpecification (0..1)` means optional (zero or one).

2.4 RateSpecification Type Definition

File: `common-domain-model-5.27.0/rosetta-source/src/main/rosetta/product-asset-type.rosetta`

Lines: 223-230

Code Evidence:

```
223|type RateSpecification: <" A class to specify the fixed interest rate, floati
ng interest rate or inflation rate.">
224|
225|   fixedRate FixedRateSpecification (0..1) <"The fixed rate or fixed rate sp
ecification expressed as explicit fixed rates and dates.">
226|   floatingRate FloatingRateSpecification (0..1) <"The floating interest rate
specification, which includes the definition of the floating rate index. the teno
r, the initial value, and, when applicable, the spread, the rounding convention,
the averaging method and the negative interest rate treatment.">
227|   inflationRate InflationRateSpecification (0..1) <"An inflation rate calculati
on definition.">
228|   condition:
229|     one-of
```

Explanation:

- Line 229: `one-of` means exactly one of `fixedRate`, `floatingRate`, or `inflationRate` must be present.
- For `CapFloor`, `floatingRate` must be present.

2.5 FloatingRateBase and FloatingRate Type Definitions

File: `common-domain-model-5.27.0/rosetta-source/src/main/rosetta/product-asset-type.rosetta`

Lines: 693-706

Code Evidence:

```
693|type FloatingRateBase: <"A class defining a floating interest rate through
the specification of the floating rate index, the tenor, the multiplier schedule, t
he spread, the qualification of whether a specific rate treatment and/or a cap
or floor apply.">
694|   [metadata key]
695|
696|   rateOption FloatingRateOption (0..1)
697|     [metadata address "pointsTo"=Observable→rateOption]
```

```

698|   spreadSchedule SpreadSchedule (0..1)
699|   capRateSchedule StrikeSchedule (0..1) <"The cap rate or cap rate sche
    |   dule, if any, which applies to the floating rate. The cap rate (strike) is only req
    |   uired where the floating rate on a swap stream is capped at a certain level. A
    |   cap rate schedule is expressed as explicit cap rates and dates and the step da
    |   tes may be subject to adjustment in accordance with any adjustments specifie
    |   d in calculationPeriodDatesAdjustments. The cap rate is assumed to be exclus
    |   ive of any spread and is a per annum rate, expressed as a decimal. A cap rate
    |   of 5% would be represented as 0.05.">
700|   floorRateSchedule StrikeSchedule (0..1) <"The floor rate or floor rate sc
    |   hedule, if any, which applies to the floating rate. The floor rate (strike) is only r
    |   equired where the floating rate on a swap stream is floored at a certain strike l
    |   evel. A floor rate schedule is expressed as explicit floor rates and dates and th
    |   e step dates may be subject to adjustment in accordance with any adjustment
    |   s specified in calculationPeriodDatesAdjustments. The floor rate is assumed t
    |   o be exclusive of any spread and is a per annum rate, expressed as a decimal.
    |   The floor rate of 5% would be represented as 0.05.">
701|
702|type FloatingRate extends FloatingRateBase:
703|   floatingRateMultiplierSchedule RateSchedule (0..1)
704|   rateTreatment RateTreatmentEnum (0..1)
705|   calculationParameters FloatingRateCalculationParameters (0..1)
706|   fallbackRate FallbackRateParameters (0..1)

```

Explanation:

- Lines 699-700: `capRateSchedule` and `floorRateSchedule` are optional (`0..1`).
- `FloatingRateSpecification` extends `FloatingRate` , which extends `FloatingRateBase` , so it inherits these fields.

2.6 StrikeSchedule Type Definition

File: `common-domain-model-5.27.0/rosetta-source/src/main/rosetta/product-template-type.rosetta`

Lines: 951-954

Code Evidence:

```

951|type StrikeSchedule extends RateSchedule: <"A class describing a sched
    |ule of cap or floor rates.">
952|
953|   buyer PayerReceiverEnum (0..1) <"The buyer of the option.">
954|   seller PayerReceiverEnum (0..1) <"The party that has sold.">

```

Explanation:

- `StrikeSchedule` extends `RateSchedule` (contains price/rate schedule data).
- Includes `buyer` and `seller` roles.

PART 3: STEP-BY-STEP EXECUTION FLOW

STEP 1: Evaluate Asset Class Qualification

Code Reference: Line 1457

```

Qualify_AssetClass_InterestRate(economicTerms) = True

```

Function Definition: Lines 24-54 of `product-qualification-func.rosetta`

Code Evidence:

```

24|func Qualify_AssetClass_InterestRate: <"Qualifies a product as having the
    |Asset Class classification Interest Rate.">
25|   inputs:
26|     economicTerms EconomicTerms (1..1)
27|   output:
28|     is_product boolean (1..1)
29|
30|   alias optionUnderlier: economicTerms → payout → optionPayout only-el
    |ement → underlier
31|   alias forwardUnderlier:
32|     economicTerms → payout → forwardPayout only-element → underlier
33|
34|   set is_product:

```

```

35|     economicTerms → payout → interestRatePayout only exists
36|     or (economicTerms → payout → optionPayout only exists
37|         and (optionUnderlier → security → securityType = SecurityTypeE
num → Debt
38|         or optionUnderlier → security → productTaxonomy → primaryA
ssetClass any = AssetClassEnum → InterestRate
39|         or optionUnderlier → index → productTaxonomy → primaryAss
etClass any = AssetClassEnum → InterestRate
40|         or if optionUnderlier exists
41|             then Qualify_AssetClass_InterestRate(
42|                 optionUnderlier → contractualProduct → economic
Terms
43|                 ) = True
44|             or Qualify_AssetClass_InterestRate(
45|                 optionUnderlier → security → economicTerms
46|                 ) = True
47|             else False))
48|     or ((economicTerms → payout → forwardPayout only exists
49|         or (economicTerms → payout → forwardPayout exists
50|             and (economicTerms → payout → interestRatePayout exists
51|                 or economicTerms → payout → cashflow exists)))
52|         and (forwardUnderlier → security → securityType = SecurityType
Enum → Debt
53|         or forwardUnderlier → security → productTaxonomy → primary
AssetClass any = AssetClassEnum → InterestRate
54|         or forwardUnderlier → index → productTaxonomy → primaryAs
setClass any = AssetClassEnum → InterestRate))

```

Explanation:

- Line 35: `only exists` means `interestRatePayout` exists and no other payout types exist.
- For CapFloor, this path typically evaluates to `True` because `interestRatePayout` exists and other payout types do not.

Example Evaluation:

```

{
  "economicTerms": {
    "payout": {
      "interestRatePayout": [ { ... } ]
      // No other payout types present
    }
  }
}

```

Result: `True` (line 35 condition satisfied)

STEP 2: Count Interest Rate Payouts

Code Reference: Line 1458

`economicTerms → payout → interestRatePayout count = 1`

Explanation:

- `>` navigates from `economicTerms` to `payout` to `interestRatePayout`.
- `count` returns the number of elements in the collection.
- `= 1` requires exactly one element.

Data Path Navigation:

1. Start: `economicTerms` (EconomicTerms instance)
2. Navigate: `economicTerms → payout` → `Payout` instance
3. Navigate: `economicTerms → payout → interestRatePayout` → `InterestRatePayout[]` (collection)
4. Apply: `count` → integer
5. Compare: `count = 1` → boolean

Example Cases:

Case A: Exactly 1 payout (QUALIFIES)

```
{
  "payout": {
    "interestRatePayout": [
      { "rateSpecification": { ... } }
    ]
  }
}
```

Evaluation: `count = 1` → `True`

Case B: Zero payouts (FAILS)

```
{
  "payout": {
    "interestRatePayout": []
  }
}
```

Evaluation: `count = 0` → `False`

Case C: Two payouts (FAILS - This would be a swap)

```
{
  "payout": {
    "interestRatePayout": [
      { "rateSpecification": { "fixedRate": { ... } } },
      { "rateSpecification": { "floatingRate": { ... } } }
    ]
  }
}
```

Evaluation: `count = 2` → `False`

STEP 3: Navigate to FloatingRate and Check Cap/Floor Schedules

Code Reference: Lines 1460-1461

economicTerms → payout → interestRatePayout → rateSpecification → floatingRate → capRateSchedule exists

or

economicTerms → payout → interestRatePayout → rateSpecification → floatingRate → floorRateSchedule exists

Explanation:

- Navigate through: economicTerms → payout → interestRatePayout → rateSpecification → floatingRate → capRateSchedule / floorRateSchedule
- exists checks presence (not null/absent).
- or means at least one must exist.

Detailed Path Breakdown:

Path 1: Checking capRateSchedule

Step 1: economicTerms

↓ (navigate to payout attribute)

Step 2: economicTerms → payout

↓ (navigate to interestRatePayout collection, get first element)

Step 3: economicTerms → payout → interestRatePayout

↓ (navigate to rateSpecification attribute)

Step 4: economicTerms → payout → interestRatePayout → rateSpecification

↓ (navigate to floatingRate attribute)

Step 5: economicTerms → payout → interestRatePayout → rateSpecification -> floatingRate

↓ (navigate to capRateSchedule attribute)

Step 6: economicTerms → payout → interestRatePayout → rateSpecification -> floatingRate → capRateSchedule

↓ (check if exists)

Step 7: exists → boolean (True if present, False if absent)

Path 2: Checking floorRateSchedule

Same path as above, but ends with:

Step 6: economicTerms → payout → interestRatePayout → rateSpecification -
> floatingRate → floorRateSchedule

Step 7: exists → boolean

Important Notes:

- `interestRatePayout` is a collection (`0..*`). The path implicitly accesses the first element or all elements.
- `rateSpecification` is optional (`0..1`). If absent, the path fails.
- `floatingRate` is optional (`0..1`). If absent, the path fails.
- `capRateSchedule` and `floorRateSchedule` are optional (`0..1`). `exists` checks for presence.

PART 4: COMPLETE LOGIC EVALUATION

4.1 Full Boolean Expression

Code Reference: Lines 1456-1461

```
set is_product:
  Qualify_AssetClass_InterestRate(economicTerms) = True
    and economicTerms → payout → interestRatePayout count = 1
      // qualifies the product as having a cap and/or floor in the interestRatePayout
    and economicTerms → payout → interestRatePayout → rateSpecification
      n → floatingRate → capRateSchedule exists
      or economicTerms → payout → interestRatePayout → rateSpecification
        floatingRate → floorRateSchedule exists
```

Operator Precedence:

1. `and` has higher precedence than `or`
2. Equivalent to:

```
(
  Qualify_AssetClass_InterestRate(economicTerms) = True
  and economicTerms → payout → interestRatePayout count = 1
  and economicTerms → payout → interestRatePayout → rateSpecification →
floatingRate → capRateSchedule exists
)
or
(
  economicTerms → payout → interestRatePayout → rateSpecification → float
ingRate → floorRateSchedule exists
)
```

Corrected Logic (with proper grouping):

The actual logic should be:

```
Qualify_AssetClass_InterestRate(economicTerms) = True
and economicTerms → payout → interestRatePayout count = 1
and (
  economicTerms → payout → interestRatePayout → rateSpecification → float
ingRate → capRateSchedule exists
  or economicTerms → payout → interestRatePayout → rateSpecification → fl
oatingRate → floorRateSchedule exists
)
```

Explanation:

- All three conditions must be true:
 1. Asset class qualification = True
 2. Exactly one interestRatePayout
 3. At least one of capRateSchedule or floorRateSchedule exists

PART 5: REAL-WORLD EXAMPLES WITH CODE EVIDENCE

Example 1: Interest Rate Cap

File: `common-domain-model-5.27.0/rosetta-source/src/main/resources/result-json-files/fpml-5-13/products/interest-rate-derivatives/ird-ex22-cap.json`


Code Evidence (Lines 49-89):

```
49|     "economicTerms" : {
50|       "payout" : {
51|         "interestRatePayout" : [ {
52|           "payerReceiver" : {
53|             "payer" : "Party1",
54|             "receiver" : "Party2"
55|           },
56|           "rateSpecification" : {
57|             "floatingRate" : {
58|               "rateOption" : {
59|                 "address" : {
60|                   "scope" : "DOCUMENT",
61|                   "value" : "rateOption-1"
62|                 }
63|               },
64|               "capRateSchedule" : {
65|                 "price" : {
66|                   "address" : {
67|                     "scope" : "DOCUMENT",
68|                     "value" : "price-1"
69|                   }
70|                 },
71|                 "buyer" : "Receiver",
72|                 "seller" : "Payer"
73|               },
74|               "meta" : {
75|                 "globalKey" : "d44f9f26"
76|               }
77|             }
78|           }
79|         ]
80|       }
81|     }
```


```
88|         }  
89|     },
```

Step-by-Step Evaluation:


Step 1: Asset Class Qualification

- Input: `economicTerms` with `interestRatePayout` present, no other payout types
- Evaluation: `economicTerms → payout → interestRatePayout only exists` → `True`
- Result: `Qualify_AssetClass_InterestRate(economicTerms) = True` 


Step 2: Count Check

- Path: `economicTerms → payout → interestRatePayout`
- Count: `1` (array has 1 element)
- Evaluation: `count = 1` → `True` 

Step 3: Cap Schedule Check

- Path: `economicTerms → payout → interestRatePayout → rateSpecification → floatingRate → capRateSchedule`
- Line 75: `"capRateSchedule"` exists
- Evaluation: `exists` → `True` 

Step 4: Floor Schedule Check

- Path: `economicTerms → payout → interestRatePayout → rateSpecification → floatingRate → floorRateSchedule`
- No `floorRateSchedule` in JSON
- Evaluation: `exists` → `False` 

Final Result:

True and True and (True or False) = True and True and True = True

Qualified as: `InterestRate_CapFloor` 

Example 2: Interest Rate Floor

File: common-domain-model-5.27.0/rosetta-source/src/main/resources/result-json-files/fpml-5-13/products/interest-rate-derivatives/ird-ex23-floor.json

Code Evidence (Lines 67-89):

```
67|         "rateSpecification" : {
68|             "floatingRate" : {
69|                 "rateOption" : {
70|                     "address" : {
71|                         "scope" : "DOCUMENT",
72|                         "value" : "rateOption-1"
73|                     }
74|                 },
75|                 "floorRateSchedule" : {
76|                     "price" : {
77|                         "address" : {
78|                             "scope" : "DOCUMENT",
79|                             "value" : "price-1"
80|                         }
81|                     },
82|                     "buyer" : "Receiver",
83|                     "seller" : "Payer"
84|                 },
```

Step-by-Step Evaluation:

Step 1: Asset Class Qualification

- Same as Example 1 → True 


Step 2: Count Check

- Same as Example 1 → True 

Step 3: Cap Schedule Check

- No `capRateSchedule` in JSON → False 

Step 4: Floor Schedule Check

- Line 75: `"floorRateSchedule"` exists → True 

Final Result:

True and True and (False or True) = True and True and True = True

Qualified as: `InterestRate_CapFloor` 

Example 3: Interest Rate Collar

File: `common-domain-model-5.27.0/rosetta-source/src/main/resources/result-json-files/fpml-5-13/products/interest-rate-derivatives/ird-ex24-collar.json`

Code Evidence (Lines 67-98):

```
67|         "rateSpecification" : {
68|             "floatingRate" : {
69|                 "rateOption" : {
70|                     "address" : {
71|                         "scope" : "DOCUMENT",
72|                         "value" : "rateOption-1"
73|                     }
74|                 },
75|                 "capRateSchedule" : {
76|                     "price" : {
77|                         "address" : {
78|                             "scope" : "DOCUMENT",
79|                             "value" : "price-1"
80|                         }
81|                     },
82|                     "buyer" : "Receiver",
83|                     "seller" : "Payer"
84|                 },
85|                 "floorRateSchedule" : {
86|                     "price" : {
87|                         "address" : {
88|                             "scope" : "DOCUMENT",
89|                             "value" : "price-2"
90|                         }
91|                     }
92|                 }
93|             }
94|         }
95|     }
96| }
97|
98| }
```

```
91|         },
92|         "buyer" : "Payer",
93|         "seller" : "Receiver"
94|     },
```

Step-by-Step Evaluation:

Step 1: Asset Class Qualification → True ✓

Step 2: Count Check → True ✓

Step 3: Cap Schedule Check

- Line 75: "capRateSchedule" exists → True ✓

Step 4: Floor Schedule Check

- Line 85: "floorRateSchedule" exists → True ✓

Final Result:

True and True and (True or True) = True and True and True = True

Qualified as: InterestRate_CapFloor ✓

PART 6: FAILURE CASES WITH EXPLANATIONS

Failure Case 1: Missing Asset Class Qualification

Example:

```
{
  "economicTerms": {
    "payout": {
      "creditDefaultPayout": { ... },
      "interestRatePayout": [ { ... } ]
    }
  }
}
```

Evaluation:

- Step 1: `Qualify_AssetClass_InterestRate(economicTerms)`
 - `interestRatePayout` only exists → `False` (because `creditDefaultPayout` also exists)
 - Result: `False` ❌
 - Final: `False and ...` → `False`
 - **Does NOT qualify** as `InterestRate_CapFloor`
-

Failure Case 2: Zero Interest Rate Payouts

Example:

```
{
  "economicTerms": {
    "payout": {
      "interestRatePayout": []
    }
  }
}
```

Evaluation:

- Step 1: Asset class → `False` (no `interestRatePayout`) ❌
 - Step 2: `count = 0` → `False` ❌
 - **Does NOT qualify**
-

Failure Case 3: Two Interest Rate Payouts (Swap)

Example:

```
{
  "economicTerms": {
    "payout": {
      "interestRatePayout": [
        { "rateSpecification": { "fixedRate": { ... } } },

```

```

    { "rateSpecification": { "floatingRate": { ... } } }
  ]
}
}
}

```

Evaluation:

- Step 1: Asset class → True ✓
- Step 2: count = 2 → False ✗
- **Does NOT qualify** (this is a swap, not a cap/floor)

Failure Case 4: Fixed Rate Instead of Floating Rate

Example:

```

{
  "economicTerms": {
    "payout": {
      "interestRatePayout": [ {
        "rateSpecification": {
          "fixedRate": { ... }
        }
      } ]
    }
  }
}

```

Evaluation:

- Step 1: Asset class → True ✓
- Step 2: count = 1 → True ✓
- Step 3: Path > floatingRate → capRateSchedule fails (no floatingRate) → False ✗
- Step 4: Path > floatingRate → floorRateSchedule fails → False ✗
- Final: True and True and (False or False) → False

- Does NOT qualify

Failure Case 5: Floating Rate Without Cap or Floor

Example:

```
{
  "economicTerms": {
    "payout": {
      "interestRatePayout": [ {
        "rateSpecification": {
          "floatingRate": {
            "rateOption": { ... },
            "spreadSchedule": { ... }
            // No capRateSchedule or floorRateSchedule
          }
        }
      } ]
    }
  }
}
```

Evaluation:

- Step 1: Asset class → True ✓
- Step 2: count = 1 → True ✓
- Step 3: capRateSchedule exists → False ✗
- Step 4: floorRateSchedule exists → False ✗
- Final: True and True and (False or False) → False
- Does NOT qualify (this is a floating rate leg, not a cap/floor)

PART 7: OPERATOR EXPLANATIONS

7.1 The **exists** Operator

Usage: `path → attribute exists`

Meaning: Returns `True` if the attribute is present (not null/absent), `False` otherwise.

Code Evidence from other functions:

```
518|     businessEvent → instruction → primitiveInstruction → observation →  
    observationEvent → creditEvent exists
```

For CapFloor:

- `capRateSchedule exists` checks if `capRateSchedule` is present in the `FloatingRate` object
- Since `capRateSchedule` is `(0..1)`, it can be absent
- `exists` returns `True` only when it's actually present

7.2 The `count` Operator

Usage: `collection count`

Meaning: Returns the number of elements in a collection.

Code Evidence:

```
24|     and closedTradeStates count = 1  
25|     and openTradeStates count >= 1
```

For CapFloor:

- `interestRatePayout count` returns the number of `InterestRatePayout` elements
- `count = 1` requires exactly one element

7.3 The `only exists` Operator

Usage: `attribute only exists`

Meaning: Returns `True` if the attribute exists and no other alternative attributes exist.

Code Evidence:

35| economicTerms → payout → interestRatePayout only exists

For Asset Class Qualification:

- `interestRatePayout only exists` means:
 - `interestRatePayout` must exist (at least one element)
 - AND no other payout types exist (`optionPayout` , `forwardPayout` , `creditDefaultPayout` , etc.)

7.4 The **>** Navigation Operator

Usage: `object → attribute`

Meaning: Navigates from an object to one of its attributes.

Examples:

- `economicTerms → payout` navigates from `EconomicTerms` to its `payout` attribute
- `payout → interestRatePayout` navigates from `Payout` to its `interestRatePayout` collection
- Chained: `economicTerms → payout → interestRatePayout` navigates through multiple levels

PART 8: COMPLETE QUALIFICATION SUMMARY

Qualification Checklist:

✓ MUST HAVE:

1. `Qualify_AssetClass_InterestRate(economicTerms) = True`
 - Typically means `interestRatePayout` exists and no other payout types
2. Exactly one `interestRatePayout` (`count = 1`)
3. `rateSpecification` must exist
4. `rateSpecification → floatingRate` must exist (not `fixedRate` or `inflationRate`)
5. At least one of:
 - `floatingRate → capRateSchedule exists` = `True` , OR

- `floatingRate → floorRateSchedule exists = True`

✗ MUST NOT HAVE:

- Zero `interestRatePayout` elements
- Two or more `interestRatePayout` elements
- `fixedRate` or `inflationRate` instead of `floatingRate`
- Missing both `capRateSchedule` and `floorRateSchedule`
- Other payout types (for the `only exists` check in asset class qualification)

This covers the step-by-step qualification flow with code evidence for each statement. Each step includes the exact code location, type definitions, and examples showing how the logic evaluates.

How a Product Gets Qualified as InterestRate_CapFloor: Summary

Overview

Qualification is inferred from the product's economic terms, not an explicit product type. The system evaluates the structure and attributes to determine if it matches the CapFloor pattern.

The Three-Part Qualification Test

A product qualifies as `InterestRate_CapFloor` only if all three conditions are true:

1. It must be an Interest Rate product (asset class qualification)
2. It must have exactly one interest rate payout leg
3. That single leg must have a floating rate with either a cap rate schedule, a floor rate schedule, or both

Part 1: Asset Class Qualification — Must Be an Interest Rate Product

First, the system checks if the product belongs to the Interest Rate asset class.

What it checks:

- The product has at least one `interestRatePayout` element
- No other payout types are present (e.g., `creditDefaultPayout`, `commodityPayout`, `performancePayout`)

Why this matters:


- Ensures the product is Interest Rate, not Credit, Equity, Commodity, etc.
- CapFloor is a subset of Interest Rate products

How it works:

- Evaluates `Qualify_AssetClass_InterestRate(economicTerms)`
- Returns `True` if `interestRatePayout` exists and no other payout types exist
- If this fails, qualification stops

Example:

```
{
  "payout": {
    "interestRatePayout": [ { ... } ]
    // No creditDefaultPayout, no commodityPayout, etc.
  }
}
```

Result: Asset class qualification = `True` 

Part 2: Single Leg Requirement — Exactly One Interest Rate Payout

The system counts how many `interestRatePayout` elements exist.

What it checks:

- `interestRatePayout count = 1`

Why this matters:

- Caps, floors, and collars have one leg
- Swaps have two legs (fixed and floating)
- FRAs may have different structures

How it works:

- Navigates: `economicTerms → payout → interestRatePayout`
- Counts elements in the collection
- Requires exactly `1`

Examples:

✅ Qualifies (1 payout):

```
{
  "interestRatePayout": [
    { "rateSpecification": { "floatingRate": { ... } } }
  ]
}
```

Count = 1 ✅

❌ Fails (0 payouts):

```
{
  "interestRatePayout": []
}
```

Count = 0 ❌

❌ Fails (2 payouts - this is a swap):

```
{
  "interestRatePayout": [
    { "rateSpecification": { "fixedRate": { ... } } },
    { "rateSpecification": { "floatingRate": { ... } } }
  ]
}
```

```
]
}
```

Count = 2 ❌

Part 3: Cap/Floor Schedule Requirement — Must Have Cap or Floor

The system checks if the single `interestRatePayout` has a floating rate with a cap schedule, a floor schedule, or both.

What it checks:

- The payout has a `rateSpecification`
- The `rateSpecification` contains a `floatingRate` (not `fixedRate` or `inflationRate`)
- The `floatingRate` contains at least one of:
 - `capRateSchedule` (exists), OR
 - `floorRateSchedule` (exists)

Why this matters:

- Caps/floors/collars are defined by cap/floor schedules on a floating rate
- A plain floating rate leg without cap/floor is not a CapFloor product

How it works:

Step 3a: Navigate to rateSpecification

- Path: `economicTerms → payout → interestRatePayout → rateSpecification`
- If missing, qualification fails

Step 3b: Verify it's a floatingRate

- Path: `rateSpecification → floatingRate`
- Must be `floatingRate` , not `fixedRate` or `inflationRate`
- If missing or wrong type, qualification fails

Step 3c: Check for capRateSchedule

- Path: `floatingRate → capRateSchedule`
- Uses `exists` to check presence
- Returns `True` if present, `False` if absent

Step 3d: Check for floorRateSchedule

- Path: `floatingRate → floorRateSchedule`
- Uses `exists` to check presence
- Returns `True` if present, `False` if absent

Step 3e: Combine with OR logic

- At least one of `capRateSchedule exists` OR `floorRateSchedule exists` must be `True`

Examples:

✅ Interest Rate Cap (capRateSchedule only):

```
{
  "rateSpecification": {
    "floatingRate": {
      "rateOption": { ... },
      "capRateSchedule": {
        "price": { ... },
        "buyer": "Receiver",
        "seller": "Payer"
      }
    }
    // No floorRateSchedule
  }
}
```

- `capRateSchedule exists` = `True` ✅
- `floorRateSchedule exists` = `False`
- Combined: `True OR False` = `True` ✅
- **Qualifies as InterestRate_CapFloor**

✓ Interest Rate Floor (floorRateSchedule only):

```
{
  "rateSpecification": {
    "floatingRate": {
      "rateOption": { ... },
      "floorRateSchedule": {
        "price": { ... },
        "buyer": "Receiver",
        "seller": "Payer"
      }
    }
    // No capRateSchedule
  }
}
```

- `capRateSchedule exists` = `False`
- `floorRateSchedule exists` = `True` ✓
- Combined: `False OR True` = `True` ✓
- **Qualifies as InterestRate_CapFloor**

✓ Interest Rate Collar (both cap and floor):

```
{
  "rateSpecification": {
    "floatingRate": {
      "rateOption": { ... },
      "capRateSchedule": { ... },
      "floorRateSchedule": { ... }
    }
  }
}
```

- `capRateSchedule exists` = `True` ✓

- `floorRateSchedule exists` = `True` ✓
- Combined: `True OR True` = `True` ✓
- **Qualifies as InterestRate_CapFloor**

✗ Plain Floating Rate (no cap/floor):

```
{
  "rateSpecification": {
    "floatingRate": {
      "rateOption": { ... },
      "spreadSchedule": { ... }
      // No capRateSchedule or floorRateSchedule
    }
  }
}
```

- `capRateSchedule exists` = `False`
- `floorRateSchedule exists` = `False`
- Combined: `False OR False` = `False` ✗
- **Does NOT qualify** (this is a floating rate leg, not a cap/floor)

✗ Fixed Rate (wrong rate type):

```
{
  "rateSpecification": {
    "fixedRate": { ... }
    // No floatingRate at all
  }
}
```

- Path to `floatingRate` fails (doesn't exist)
- Cannot check cap/floor schedules
- **Does NOT qualify**

The Complete Evaluation Flow

When `Qualify_InterestRate_CapFloor` is called:

1. **Start:** Receive `economicTerms` as input
 2. **Step 1:** Call `Qualify_AssetClass_InterestRate(economicTerms)`
 - If `False` → Stop, return `False`
 - If `True` → Continue
 3. **Step 2:** Count `interestRatePayout` elements
 - If `count ≠ 1` → Stop, return `False`
 - If `count = 1` → Continue
 4. **Step 3:** Navigate to `floatingRate` within the single payout
 - If path fails (missing `rateSpecification` or `floatingRate`) → Stop, return `False`
 - If path succeeds → Continue
 5. **Step 4:** Check for cap/floor schedules
 - Check `capRateSchedule exists`
 - Check `floorRateSchedule exists`
 - If both are `False` → Stop, return `False`
 - If at least one is `True` → Continue
 6. **Final:** All conditions met → Return `True` and qualify as `InterestRate_CapFloor`
-

Why This Design?

- Structure-based: No explicit product type field; qualification is inferred
 - Composable: Multiple qualification functions can match the same product
 - Taxonomy-aligned: Maps to ISDA Taxonomy v1 and v2
 - Deterministic: Same structure always produces the same result
-

Key Distinctions

CapFloor vs. Interest Rate Swap:

- CapFloor: 1 `interestRatePayout` with cap/floor schedules
- Swap: 2 `interestRatePayout` elements (typically fixed and floating)

CapFloor vs. Plain Floating Rate Leg:

- CapFloor: Has `capRateSchedule` or `floorRateSchedule`
- Plain floating rate: No cap/floor schedules

CapFloor vs. Fixed Rate Product:

- CapFloor: Uses `floatingRate`
 - Fixed rate: Uses `fixedRate`
-

Final Summary

A product qualifies as `InterestRate_CapFloor` when:

1. It is an Interest Rate product (has `interestRatePayout` and no other payout types)
2. It has exactly one `interestRatePayout` leg
3. That leg uses a `floatingRate` with at least one of `capRateSchedule` or `floorRateSchedule` present

All three conditions must be true. The system navigates the data structure, checks each condition, and returns `True` only if all are satisfied. This identifies Interest Rate Caps, Floors, and Collars as `InterestRate_CapFloor` according to ISDA taxonomy standards.