# Rosetta Expressions

1. Documentation: [https://docs.rosetta-technology.io/rosetta/rosetta-dsl/rosetta-modelling-component/#expression-component](https://docs.rosetta-technology.io/rosetta/rosetta-dsl/rosetta-modelling-component/%23expression-component)
2. Constant Expression
3. Path Expression
   1. conditional statements: if, then, else
   2. comparison operators: =, <>, <, <=, >=, >
   3. boolean operators: and, or
   4. arithmetic operators: +, -, \*, /

# Condition statement

condition <ConditionName>: (optional: <"Description">)

<booleanOperator>

## Choice condition

**<**choiceType**>** choice **<**attribute1**>**, **<**attribute2**>**, **<...>**

## one-of

special case where a required choice logic applies to *all* the attributes of a given type, resulting in one and only one of them being present in any instance of that type

# Functions found in type conditions

## FilterPartyRole

func FilterPartyRole: <"Filters the list of partyRoles based on the provided partyRoleEnum.">

    inputs:

        partyRoles PartyRole (0..\*)

        partyRoleEnum PartyRoleEnum (1..1)

    output:

        filteredPartyRoles PartyRole (0..\*)

    add filteredPartyRoles:

        partyRoles

            filter [ item -> role = partyRoleEnum ]

## FpmlIrd8

func FpmlIrd8: <"FpML validation rule ird-8 - If the same party is specified as the payer and receiver, then different accounts must be specified.">

    inputs:

        tradableProduct TradableProduct (1..1)

        accounts Account (0..\*)

    output:

        success boolean (1..1) <"Validation result">

Chosen Java implementation to review functional expression:

@ImplementedBy(FpmlIrd8.FpmlIrd8Default.class)  
public abstract class FpmlIrd8 implements RosettaFunction {  
 public FpmlIrd8() {  
 }  
  
 public Boolean evaluate(TradableProduct tradableProduct, List<? extends Account> accounts) {  
 Boolean successHolder = this.doEvaluate(tradableProduct, accounts);  
 Boolean success = this.assignOutput(successHolder, tradableProduct, accounts);  
 return success;  
 }  
  
 private Boolean assignOutput(Boolean success, TradableProduct tradableProduct, List<? extends Account> accounts) {  
 return success;  
 }  
  
 protected abstract Boolean doEvaluate(TradableProduct var1, List<? extends Account> var2);  
  
 public static final class FpmlIrd8Default extends FpmlIrd8 {  
 public FpmlIrd8Default() {  
 }  
  
 protected Boolean doEvaluate(TradableProduct tradableProduct, List<? extends Account> accounts) {  
 return null;  
 }  
 }  
}

## FilterOpenTradeStates

func FilterOpenTradeStates: <"Filter to only 'open' TradeState - where both the closedState and positionState are not set.">

    inputs:

        tradeStates TradeState (0..\*)

    output:

        openTradeStates TradeState (0..\*)

    add openTradeStates:

        tradeStates

            filter [ item -> state -> closedState is absent ]

## Qualify\_AssetClass\_Equity

func Qualify\_AssetClass\_Equity: <"Qualifies a product as having the Asset Class classification Equity.">

    inputs: underlier Product (1..1)

    output: is\_product boolean (1..1)

    set is\_product:

        underlier -> security -> securityType = SecurityTypeEnum -> Equity

            or (underlier -> security -> securityType = SecurityTypeEnum -> Fund

                and underlier -> security -> fundType = FundProductTypeEnum -> ExchangeTradedFund)

            or (underlier -> security -> securityType = SecurityTypeEnum -> Fund

                and underlier -> security -> fundType = FundProductTypeEnum -> MutualFund)

            or underlier -> security -> securityType = SecurityTypeEnum -> Warrant

            or (underlier -> index exists /\* and underlier -> index -> indexType = IndexTypeEnum -> Equity \*/) // No FpML mappings exist for IndexTypeEnum

            // Qualifies that the underlier is a basket composed of equity products only

            or (underlier -> basket exists

                and (underlier -> basket -> basketConstituent -> security -> securityType any = SecurityTypeEnum -> Equity

                    or (underlier -> basket -> basketConstituent -> security -> securityType any = SecurityTypeEnum -> Fund

                        and underlier -> basket -> basketConstituent -> security -> fundType any = FundProductTypeEnum -> ExchangeTradedFund)

                    or (underlier -> basket -> basketConstituent -> security -> securityType any = SecurityTypeEnum -> Fund

                        and underlier -> basket -> basketConstituent -> security -> fundType any = FundProductTypeEnum -> MutualFund)

                    or underlier -> basket -> basketConstituent -> security -> securityType any = SecurityTypeEnum -> Warrant

                    or (underlier -> basket -> basketConstituent -> index exists /\* and underlier -> basket -> basketConstituent -> index -> indexType = IndexTypeEnum -> Equity \*/)

                )

            )

## PriceQuantityTriangulation

func PriceQuantityTriangulation: <"Defines all the scenarios which triangulation can be helpful validation between Prices and Quantities.">

    inputs:

        tradeLots TradeLot (0..\*)

    output:

        success boolean (1..1)

    set success:

        if tradeLots count > 0 then

            tradeLots

                map [

                    if item -> priceQuantity -> price -> priceExpression -> priceType contains PriceTypeEnum -> CashPrice then

                        CashPriceQuantityNoOfUnitsTriangulation( item -> priceQuantity -> quantity, item -> priceQuantity -> price)

                    else True

                ]

                all = True

        else True

## CashPriceQuantityNoOfUnitsTriangulation

func CashPriceQuantityNoOfUnitsTriangulation: <"Data rule to check the relationship between cash price, notional and number of units.">

    inputs:

        quantity Quantity (0..\*)

        price Price (0..\*)

    output:

        success boolean (1..1)

    alias notional:

        FilterQuantityByCurrencyExists( quantity )

            map [ item -> amount ]

            distinct

            only-element

    alias noOfUnits:

        FilterQuantityByFinancialUnit( quantity, FinancialUnitEnum -> Share )

            map [ item -> amount ]

            only-element

    alias cashPrice:

        price -> amount only-element

    set success:

        if cashPrice exists and noOfUnits exists and notional exists

        then cashPrice \* noOfUnits = notional

## FilterQuantityByCurrencyExists

func FilterQuantityByCurrencyExists: <"Filter list of quantities based on unit type.">

    inputs:

        quantities Quantity (0..\*) <"List of quantities to filter.">

    output:

        filteredQuantities Quantity (0..\*)

    add filteredQuantities:

        quantities

            filter [ item -> unitOfAmount -> currency exists ]

## FilterQuantityByFinancialUnit

func FilterQuantityByFinancialUnit: <"Filter list of quantities based on unit type.">

    inputs:

        quantities Quantity (0..\*) <"List of quantities to filter.">

        financialUnit FinancialUnitEnum (1..1) <"FinancialUnitEnum unit type.">

    output:

        filteredQuantities Quantity (0..\*)

    add filteredQuantities:

        quantities

            filter [ item -> unitOfAmount -> financialUnit = financialUnit ]