Table of Contents

Manuals, Rule Language

Rule Condition

The formal definition of the 'RuleCondition' in the BNF notation is:

```
PredicateExpression ()
  (
     "OR" PredicateExpression()
  |
     "AND" PredicateExpression()
  )*
```

In other words the 'RuleCondition' represents one or more `Predicate expressions` connected with `AND` or `OR` keywords.

The Predicate expression can be either of the following:

1. Relational expression
2. String expression
3. Aggregate expression
4. List expression
5. Range expression
6. `WAS CHANGED` expression
7. `IS DEFINED` expression
8. `IS NEW` expression

All of the above expressions can be optionally negated by using the NOT keyword in front of the expression (with the expression enclosed in brackets), for example `NOT (Account.State IN 'OPEN', 'CLOSED', 'SUSPENDED')` – checks if the state of an account is not `OPEN`, `CLOSED` or `SUSPENDED`.

Arithmetic Operation

Before describing various predicate expressions we will describe another language construct that is used in most of these expressions – `Arithmetic Operation`.

`Arithmetic Operation` can be either of the following:

1. Attribute Identifier
2. Literal
3. Function
4. Aggregate Operation

Attribute Identifier

The Attribute Identifier construct is used to refer to both business objects and their attributes.

Attributes of business objects are separated from the business objects' names by the dot symbol, for example `Account.State`. IF an attribute refers to another business object then an attribute of the referred business object can be indicated using the name of the reference attribute of the parent and the name of the attribute of the referred business object, for example, `Account.Type.Name`. The level of nesting of attributes has no limit.

Literal

Literal represents a constant. The following types of constants are recognized:

1. `Number` (integer or floating point), for example, 3 or 5.6
2. `String` (must be enclosed in apostrophe), for example `'CLOSED'` or `'Balance must be positive'`
3. `Date` (in `dd/mm/yy` format), for example 05/12/98
4. `Date/Time` (in `dd/mm/yy HH:mm` format), for example 05/12/98 17:05
5. `Duration` (in #w#dHH:mm, where # stands for any digit, HH – for hours and mm - for minutes), for example, 2w3d10:45
6. `UNDEFINED`, for example `Loan.Item = UNDEFINED`

Function

A function performs some calculation and returns the result as a Literal. There are a number of built-in functions that ***Aware IM*** supports. New functions can be plugged in as well (see "Aware IM Programmer's Referefence"). A function may or may not have parameters. Parameters of a function must be Arithmetic Operations. IF a function does not have parameters then it can be referred to by name only. IF a function has parameters, they are listed after the function's name and enclosed in brackets. Parameters are separated by the comma symbol. Examples of functions:

- `CURRENT_DATE` - function with no parameters
- `LENGTH (Customer.Name)` – function with one parameter
- `MONTH_DIFFERENCE (Account.OpeningDate, Account.ClosingDate)` – function with two parameters

The complete list of built-in functions supported by ***Aware IM*** is provided in the here.

Aggregate Operation

The `Aggregate Operation` performs arithmetic calculations on a number of business objects and/or their attributes, for example, calculates the sum total of a certain attribute:

`SUM Account.Balance` - calculates the total balance of all available accounts.

The operation may be applied not on all available objects of the specified type, but on objects that meet the specified condition. The condition is optionally indicated after the WHERE keyword and must be enclosed in brackets, for example:

`SUM Account.Balance WHERE (Account.State='OPEN')` – calculates the total balance of all open accounts. The format of the condition, that follows the `WHERE` keyword is the same as that of

any Rule Condition.

The following Aggregate Operations are supported:

1. SUM – calculate the sum total of an attribute
2. COUNT – calculate the number of available objects
3. MIN – calculate minimum value of an attribute
4. MAX – calculate maximum value of an attribute
5. AVG – calculate average value of an attribute

note

SUM, MIN, MAX, AVG calculations must not use attributes of the Reference type, for example the following expression is not valid:
`SUM Account.Transactions.Amount`
IF aggregate of the referred object is required as in the example above, the following expression must be used:
`SUM Transaction.Amount WHERE (Transaction IN Loan.Transactions)`

note

COUNT operation MUST only use the name of the business object, which it counts, for example,

```
COUNT Account or COUNT Account WHERE (Account.State='Open')
```

.

note

Using references is not allowed, for example the following construct is invalid:

```
COUNT Account.Transactions
```

The valid expression that achieves the desired result is:

```
COUNT Transaction WHERE (Transaction IN Account.Transactions)
```

Relational expression

The Relational Expression compares two Arithmetic Operations using the following comparison operators:

1. =

2. <
3. >
4. <=
5. >=
6. <> (not equal)

Examples of valid relational expressions:

```
Account.State = 'CLOSED'
```

```
Account.Balance < Account.Type.MinBalance + 100
```

String expression

The String Expression has the following format:

ArithmeticOperation() (STARTSWITH | ENDSWITH | CONTAINS) ArithmeticOperation()

It checks whether an arithmetic operation (usually an attribute of a business object) starts with (or ends with or contains) the specified arithmetic operation that produces a string (usually a string literal), for example:

```
Account.Name STARTSWITH 'John'
```

```
Account.Name ENDSWITH 'Smith'
```

```
Account.Name CONTAINS 'it'
```

Aggregate expression

Two forms of the `Aggregate Expression` are supported in the Rule Language:

1. EXISTS expression is very similar to Aggregate Operation, except that the result of the EXISTS expression is true or false, rather than arithmetic value. The syntax and usage are the same as that of the `Aggregate Operation`. The EXISTS operation checks whether there are any instances of the specified business object in the system, for example,
    - ```
      IF EXISTS Account THEN ...
      ```
    - ```
      IF EXISTS Account WHERE (Account.State = 'Open') THEN ...
      ```
2. IN expression checks whether a particular instance of a business object is in the reference list of another object. For example,
    - ```
      IF Transaction IN Account.Transactions THEN ...
      ```
      ```

It is possible to use a special expression LOGGED_IN_USERS instead of the list of references. This will check if the specified user is logged in, for example:

```
FIND SystemUser WHERE SystemUser.LoginName='John'
```

```
IF SystemUser IN LOGGED_IN_USERS THEN DISPLAY MESSAGE 'John is logged in'
```

List expression

The List Expression checks whether the value of a particular attribute of a business object is equal to one of the specified values in the provided list. The syntax in the notation is as follows:

AttributeIdentifier () IN Literal () (, Literal ())*

In other words an attribute identifier is followed by the keyword IN and then is followed by one or more literals. Any kind of literal is valid. Some examples of valid list expressions:

```
Account.State IN 'Open', 'Closed'
```

```
Account.Balance IN 1000, 2000, 3000
```

Range expression

The Range Expression checks whether the value of a particular attribute of a business object is within the specified range of values. The syntax in the BNF notation is as follows:

ArithmeticOperation () BETWEEN ArithmeticOperation () AND ArithmeticOperation ()

Note that using comma instead of AND is also allowed. Examples of valid range expressions:

```
Account.Balance BETWEEN 10000 AND 20000;
```

```
Transaction.Amount BETWEEN Account.Balance/2, 5000
```

> **note**
> both ranges are inclusive

WAS CHANGED expression

The WAS CHANGED expression checks whether the value of a particular attribute of a business object has been changed compared to the value stored in the system. There are several variations of this expression:

1. AttributeIdentifier () `WAS CHANGED` – checks if an attribute has been changed. For example,

   ```
   IF Account.State WAS CHANGED THEN ...
   ```

   The entire object can be checked as well – in this case all attributes of the object are checked for changes, for example

   ```
   IF Account WAS CHANGED
   ```

2. AttributeIdentifier () `WAS CHANGED TO` Literal () – checks if an attribute has been changed and the new value is equal to the specified value, for example,

   ```
   IF Account.State WAS CHANGED TO 'Open' THEN ...
   ```

3. AttributeIdentifier () `WAS CHANGED BY ANY USER` - deprecated. Functionally equivalent to `WAS CHANGED`.

> **note**
>
> When comparing new and old values of the attribute, the WAS CHANGED expression only checks whether the value has been changed compared to the value of the attribute in the *last stable version* of the business object. See the "Evaluation of WAS CHANGED expressions" section for a more detailed explanation.

Expressions that track changes in a list

The `WAS CHANGED` expression can be used for reference lists as well as for ordinary attributes. The `WAS CHANGED` expression for lists indicates whether there were any references changed or removed from the list compared to the last stable version (see the "Evaluation of WAS CHANGED expressions" section for a more detailed explanation). For example,

```
IF Account.Transactions WAS CHANGED THEN ...
```

It is possible to identify more precisely how a reference list has been changed and perform actions based on the values of the objects that have been added or removed from the list. The `WAS ADDED TO` expression can be used to check whether any objects have been added to the list and the `WAS REMOVED FROM` expression can be used to check whether the objects have been removed from the list. To refer to the objects that have been added or removed, the Added and Removed instance prefixes can be used respectively (see the "Instance Prefixes" section. For example,

```
IF Transaction WAS ADDED TO Account.Transactions THEN
    INCREASE Account.Balance BY AddedTransaction.Amount
```

```
IF Transaction WAS REMOVED FROM Account.Transactions THEN
    REDUCE Account.Balance BY RemovedTransaction.Amount
```

IF the list itself hasn't changed, but an element belonging to the list has been, then this situation can be checked using the following expression:

```
IF Transaction FROM Account.Transactions WAS CHANGED THEN
    INCREASE Account.Balance BY (ChangedTransaction.Amount –
OLD_VALUE(Transaction.Amount))
```

IS UNDEFINED expression

The `IS UNDEFINED` expression checks whether the value of a particular attribute of a business object is defined. Typically a value is undefined if it is “blank” – for example, the user didn’t fill in the value in the form of the business object. For reference attributes “undefined” means that the reference list is empty. There are two variations of this expression:

1. AttributeIdentifier () `IS UNDEFINED` – checks whether an attribute is undefined, for example,

   ```
   IF Account.State IS UNDEFINED THEN ...
   ```

2. AttributeIdentifier () `IS DEFINED` – checks whether an attribute is defined, for example,

   ```
   IF Account.State IS DEFINED THEN ...
   ```

IS NEW expression

The `IS NEW` expression checks whether the particular object specified in the expression is being created, for example

```
IF NOT (Message IS NEW) THEN PROTECT Message.Subject
```

In this example the `Subject` attribute of the `Message` object is protected if the object already exists in the system.