Aware IM

Version 8.4

Rule Language Reference

CONTENTS

About this document

This document provides a formal definition of the Aware IM Rule Language. The rule language is used to specify rules, queries, tags and other configuration elements. The rule language is simple and intuitive and, unlike traditional programming languages does not require special software development skills.

1 Conventions

1. Everywhere in this document "business object" also implies "business object groups", unless explicitly stated otherwise.
2. The syntax of the Rule Language expressions is specified using the BNF notation. Some of the constructs of the BNF notation are explained below:
   a. "|" symbol indicates a choice of several keywords or expressions. For example, the expression
   `("INCREASE" | "REDUCE") AttributeIdentifier()`
   indicates that either the `INCREASE or REDUCE` keyword may precede the `Attribute Identifier`.
   b. `(some expression)*` indicates zero or more. For example the following expression,
   `PredicateExpression () ("AND" PredicateExpression())*`
   indicates that there is at least one `PredicateExpression` followed by zero or more combinations of `"AND" PredicateExpression` constructs.
   c. `[some expression]` indicates that the expression in square brackets is optional. For example the expression,
   `SEND Id() TO AttributeIdentifier() [VIA ChannelName]`
   indicates that the `VIA` keyword followed by the channel name may or may not be used.

2 Rule Definition

Rule is defined as follows:

```
If (RuleCondition) Then
     ActionList
Else
     ActionList
```

where `RuleCondition` defines one or more conditions that have to hold true before the specified `ActionList` is executed. `If` and `Else` statements are optional, so it is possible to have a rule which only has action statement(s) – such rules are called unconditional rules. It is also possible to include If statement within Else:


```
If (RuleCondition) Then
     ActionList
Else If (RuleCondition) Then
     ActionList
Else
     ActionList
```

Rules operate on business objects and/or their attributes, i.e. both the condition and action parts of a rule may refer to a business object and its attribute. Business objects are referred to by name, for example `Account`. Attributes of a business object are referred to by the attribute name where the name of the attribute is separated from the name of the business object by the dot symbol, for example `Account.State`.

Examples of rules:

- `If Account.State='CLOSED' AND Account.Balance<>0 Then REPORT ERROR 'Account with non-zero balance cannot be closed'`
- `CreditAccount.AvailableFunds=CreditAccount.CreditLimit+CreditAccount.Balance-CreditAccount.NonClearedFunds`
- ```
  If Account.Type IS UNDEFINED Then
        FIND AccountType WHERE AccountType.Name='Credit'
        Account.Type=AccountType
  ```

3 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`.

## 3.1 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`

### 3.1.1 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.

### 3.1.2 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`

### 3.1.3 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 Aware IM User Guide.

### 3.1.4 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

**Notes:**
- `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)
```


- `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').
```

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)
```

3.2 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`

3.3 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'`

3.4 Aggregate expression

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

1. `EXISTS` expression is very similar to `Aggregate Operation` (see 3.1.4), 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'
```

3.5 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 mentioned in 3.1.2 is valid. Some examples of valid list expressions:

```
Account.State IN 'Open', 'Closed'
Account.Balance IN 1000, 2000, 3000
```

3.6 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

3.7 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 in the "Aware IM User Guide" for a more detailed explanation.

3.7.1 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 in the "Aware IM User Guide" 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 in the "Aware IM User Guide"). 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))
```

3.8 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…

3.9 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.

4 Action List

The following types of actions can be executed in the Action List part of a rule:

1. Modify attribute action
2. Predefined system action
3. Process call action

4.1 Modify attribute action

The `Modify attribute` action modifies the value of a particular attribute of a business object. There are two variations of this action.

4.1.1 Assignment action.

This action assigns a new value to an attribute. The syntax is as follows:

```
AttributeIdentifier () = ArithmeticOperation ()
```

For example:

- `Account.State = 'Open'`
- `Account.Balance = 1000 + Transaction.Amount`

It is possible to assign references or clear references (set `UNDEFINED` value), for example:

```
Loan.Item = Item
Member.Loans = UNDEFINED
```

4.1.2 Incremental action

The action increments or decrements the value of a business object's attribute. The syntax is as follows:

```
(INCREASE | REDUCE) AttributeIdentifier () BY ArithmeticOperation
()
```

For example:

- `INCREASE Account.Balance BY 1000`
- `REDUCE Account.Balance BY Transaction.Amount`

Note that Incremental action allows usage of percents in arithmetic operations, for example

`INCREASE Account.Balance BY 10%` or
`REDUCE Account.Balance BY 5%`

4.2 Predefined system actions

There are a number of actions that call predefined services of the system.

4.2.1 `INSERT` action

This action inserts an instance of a business object into a reference list of another object. The syntax is as follows:

```
INSERT AttributeIdentifier() IN AttributeIdentifier()
```

where the first Attribute Identifier indicates the object to insert and the second one – the list to insert into. For example,

```
INSERT Transaction IN Account.Transactions
```

4.2.2 `REMOVE` action

This action removes an instance of a business object from a reference list of another object. The syntax is as follows:

```
REMOVE AttributeIdentifier() FROM AttributeIdentifier()
```

where the first `Attribute Identifier` indicates the object to remove and the second one – the list to remove from. For example,

```
REMOVE Transaction FROM Account.Transactions
```

4.2.3 `REPLACE` action

This action replaces an instance of a business object from a reference list of another object with some other instances. The syntax is as follows:

```
REPLACE AttributeIdentifier() IN AttributeIdentifier() WITH
AttributeIdentifier()
```

where the first `Attribute Identifier` indicates the object to replace, the second one – the list to remove from and the third one the object to be inserted. This action can be especially useful when combined with the `This` and `That` prefixes (see also “Instance Prefixes” section in the “Aware IM User Guide”). For example,

```
REPLACE ThisTransaction IN Account.Transactions WITH
ThatTransaction
```

4.2.4 `CREATE` action

This action creates an instance of the specified business object. The syntax is:


```
CREATE Id() or CREATE ArithmeticOperation() [NO VALIDATION][NO
RULES]
```

where `Id()` is identifier of the business object to create, for example

```
CREATE Account
```

and `Arithmetic Operation` is an expression producing a string indicating the name of the object to create.

Note that the identifier here is not the attribute identifier and thus may not indicate a reference. The action is also not applicable to business object groups.

`CREATE` action can optionally initialize one or more attributes of the business object that it creates, for example:

```
CREATE Account WITH Account.State='Open', Account.Balance=0
```

The attributes to be initialized must follow the `WITH` keyword and must be separated by comma. Any arithmetic operation is valid as the initialization expression.

Sometimes it may be necessary to create several instances of a business object with a single `CREATE` action. The following constructs are supported:

1. `CREATE Id() FOR EACH AttributeIdentifier()`

This action will create as many instances of the object as there are instances of another business object in the Context. For example,

```
CREATE Transaction FOR EACH Account
```

2. `CREATE Id() FOR EACH (DAY | WEEK | MONTH | QUARTER | YEAR | WEEK_DAY | WEEKEND_DAY) BETWEEN ArithmeticOperation() AND ArithmeticOperation`

This action will create as many instances of the specified business object as there are days(weeks/months/quarters/years) in the specified date interval. Note that the Arithmetic Operations here must be operations on dates, for example,

```
CREATE Transaction FOR EACH DAY BETWEEN Account.OpeningDate AND
Account.ClosingDate
```

The date interval includes both starting and ending dates. Note also that if the object being created has any attributes of the Date type, these attributes can be initialized with the value of the date for which the object is being created. For example,


```
CREATE Transaction FOR EACH DAY BETWEEN Account.OpeningDate AND
Account.ClosingDate WITH Transaction.AppliedDate=Day
```

Every transaction created in this way will have the `AppliedDate` attribute initialized to the day for which the transaction is created.

3. `CREATE Id() FOR EACH NUMBER BETWEEN ArithmeticOperation() AND ArithmeticOperation`

This action will create as many instances of the specified business object as there are numbers in the specified interval. Note that the Arithmetic Operations here must be operations on numbers, for example,

```
CREATE Transaction FOR EACH NUMBER BETWEEN 1 AND 3
```

The interval includes both starting and ending numbers.

4. `CREATE Id() FOR EACH FILE IN ArithmeticOperation()`

This action will create as many instances of the specified business object as there are files in the specified directory. The Arithmetic Operation used in the expression must produce a string specifying the directory in which the system will look for files. For example,

```
CREATE Attachment FOR EACH FILE IN 'c:/mydirectory'
```

Note also that if the object being created has any attributes of the Document type, these attributes can be initialized with the file for which the object is being created. For example,

```
CREATE Attachment FOR EACH FILE IN 'c:/mydirectory' WITH
Attachment.Document=File
```

Every attachment object created in this way will have the `Document` attribute initialized to the file for which the attachment is being created.

5. Variation of the above expression is:

```
CREATE Id() FOR EACH NEW FILE IN ArithmeticOperation()
```

The only difference with the previous expression is that the file in the specified directory is deleted after the business object has been created.

The `NO VALIDATION` clause creates an object without executing validation rules (rules that `REPORT ERROR`). This can be useful if you are creating just a template of the object providing all required values later.

The `NO RULES` clause creates an object without executing any rules at all. Be careful with this clause as this may create an instance of the object with invalid data.

4.2.5 `DUPLICATE` action

This action duplicates an instance of the specified business object or list. The syntax is:

```
DUPLICATE AttributeIdentifier()
[ TO AttributeIdentifier() ]
[ EXCEPT Id () ("," Id ())* ]
[ NO RULES ]
```

Where the first attribute identifier identifies an object or list to duplicate and the second one – the list to duplicate the original list to. The `EXCEPT` clause identifies attribute names that should be excluded from duplication.

The `NO RULES` clause creates an object without executing any rules at all. Be careful with this clause as this may create an instance of the object with invalid data.

4.2.6 `DELETE` action

This action deletes all instances of the specified business object, which exist in the Context. The syntax is:

```
DELETE AttributeIdentifier()
```

where AttributeIdentifier identifies an object to be deleted.

For example,

```
DELETE Account
DELETE Account.Owner
```

4.2.7 `CLEAN` action

The CLEAN action is similar to the DELETE action in that it deletes instances of the specified business object. Unlike the DELETE action the CLEAN action does not invoke execution of rules – records are deleted in the database directly. CLEAN action is much more efficient than the DELETE action, however, you should use this action with care because it does not check the integrity of the data – if there are instances of the business objects left that refer to the deleted instances the data will be inconsistent. It is your responsibility to make sure that the data remains consistent after deletion.

The syntax of the CLEAN action is similar to that of the FIND action. Examples:

```
CLEAN ALL Account
```

The above action deletes all records of the Account object in the database.


```
CLEAN Account WHERE Account.Name='Smith'
```

The above action deletes all accounts with the name "Smith"

NOTE: The CLEAN action is not supported for business object groups. You cannot use references in the condition of this action either.

4.2.8 DELETE FILE action

This action deletes a file or a directory (and all files in this directory). The syntax is:

```
DELETE FILE ArithmeticExpression()
```

where ArithmeticExpression identifies a file or directory to be deleted.

For example,

```
DELETE FILE 'c:/temp/myfile.txt'
DELETE FILE Account.FileName
```

4.2.9 COPY FILE action

This action copies a file to a new location. The syntax is:

```
COPY FILE ArithmeticExpression() TO ArithmeticExpression()
```

where the first ArithmeticExpression identifies a file to be copied and the second one – the target location and name.

For example,

```
COPY FILE 'c:/temp/myfile.txt' TO 'c:/temp2/mynewfile.txt'
```

4.2.10 MAKE DIRECTORY Action

The MAKE DIRECTORY action creates the specified directory. The syntax is
`MAKE DIRECTORY ArithmeticExpression()`

where ArithmeticExpression identifies a directory to be created.

For example,

```
CREATE DIRECTORY 'c:/temp/mydirectory'
CREATE DIRECTORY Account.DirectoryName
```


4.2.11 SEND action

This action allows sending a notification to a particular notification receiver. The syntax is:

```
SEND Id() TO AttributeIdentifier () [ VIA ChannelName ] [ USING
AttributeIdentifier () ]
```

For example,

```
SEND ReservationOfferEmail TO Reservation.Member
```

Note that the receiver of the notification must be an intelligent business object with active communication channels.

The `VIA` keyword can be used to indicate the specific channel through which the notification will be sent, for example

```
SEND ReservationOfferNotification TO Reservation.Member VIA Email
```

If the `VIA` keyword is not specified, the default channel of the business object is used.

The `USING` keyword can be used to send emails using a particular email account, for example:

```
SEND ReservationOfferNotification TO Reservation.Member USING
EmailAccount
```

Here the instance of the EmailAccount object is taken from the Context and it should contain email account values to use when sending an email (host server, port, authentication details etc). Therefore, the object representing an account must have certain attributes defined. The definition of such an object can be created in the Configuration Tool using the "Add Outgoing Email Account" command.

When processing incoming emails (in the rules attached to the `IncomingEmail` notification), the reply can be sent to the predefined business object called `EmailSender`.

For example,

```
SEND Reply TO EmailSender
```

4.2.12 REQUEST SERVICE action

This action makes it possible to use a service of an intelligent business object. The syntax is:

```
REQUEST SERVICE Id() OF AttributeIdentifier()[ VIA ChannelName]
```


where `Id()` indicates the name of the service and `AttributeIdentifier` indicates the business object which provides the service. For example,

```
REQUEST SERVICE RegisterProduct OF Awaresoft
```

It is possible to indicate that the service should be run asynchronously – that is, if a system is executing an ordered collection of rules, one of which calls the service asynchronously, the next rule will execute immediately without waiting for the service to finish. To indicate that the service should run asynchronously, add `NOWAIT` keyword after the service. For example,

```
REQUEST SERVICE RegisterProduct OF Awaresoft NOWAIT
```

The `VIA` keyword can be used to indicate specific channel through which the service is requested, for example

```
REQUEST SERVICE RegisterProduct OF Awaresoft VIA Sockets
```

If the `VIA` keyword is not specified, the default channel of the service provider is used.

4.2.13 `REPORT ERROR` action

This action causes the system to stop executing rules and issue the specified error message to the original requestor. The syntax is:

```
REPORT ERROR ArithmeticOperation()
```

where the `Arithmetic Operation` must be the operation producing a string (usually just a string literal), for example,

```
REPORT ERROR 'Value of the attribute is not defined'
```

Note: the action stops execution of any rules caused by the initial request to the system(unless there are process failure rules defined – see further notes). If the request was issued by the user the specified error message is displayed on the user's screen. If the request did not originate from the user interface (for example, the request originated from a scheduled process) the error message is written into the log and the current process is terminated.

Note: The request that eventually caused the `REPORT ERROR` action could have changed a number of objects and attributes prior to the execution of the `REPORT ERROR` action. All changes to business objects and their attributes prior to the `REPORT ERROR` action within the context of the request are discarded when the `REPORT ERROR` action is triggered). See also the "Rules and Transactions" section in the "Aware IM User Guide".

Note: If evaluation of rules that caused execution of the `REPORT ERROR` action was initially triggered by a process and this process had failure rules attached to it, the changes to the business objects and their attributes prior to the `REPORT ERROR` action are not discarded. Instead process

failure rules are evaluated and the execution of the process continues – see also the "Process Failure Rules" section in the "Aware IM User Guide".

4.2.14 `PROTECT` action

This action protects a business object or its attribute from access by all or specified users. There are several variations of this action:

1. `PROTECT AttributeIdentifier () FROM Identifier () (AND Identifier ())*`

protects business object or attribute from access by users operating at the specified access level(s)

2. `PROTECT AttributeIdentifier () FROM ALL`

protects business object or attribute from all users

3. `PROTECT AttributeIdentifier () FROM ALL EXCEPT Identifier () (AND Identifier ())*`

protects business object or attribute from all users except those operating at the specified access level(s)

Examples:

- `If Transaction.State='APPLIED' Then PROTECT Transaction FROM ALL` – do not allow any changes to a transaction if it is in the Applied state
- `If Transaction.State='APPLIED' Then PROTECT Transaction FROM ALL EXCEPT Administrator`
- `If Account.State='CLOSED' Then PROTECT Account.Name FROM User`

If an attribute of a business object is protected it is read-only on forms of this business object. If the entire object is protected all its attributes are read-only.

Notes:

- By default protection applies not only to changes done via the user interface, but also to changes done by a process. It is possible to distinguish between these two situations by specifying `System` as access level, for example:

```
If Account.STATE <> 'NEW' THEN PROTECT Account.Balance FROM ALL EXCEPT System
```

In this case users will not be able to change the account balance when the state of the account is not new, however, it is possible to configure process rules that change the account balance.

- Protecting attributes of referred objects is not allowed, for example the action `PROTECT Transaction.Account.State FROM ALL` is invalid.

- The action can only be used in rules attached to a business object to be protected. The action may not be used in a process.
- The action specified in the format described above protects against changes to a business object or its attribute ("write protection"). If "read protection" is required, the `READ` prefix should be added in front of the action. For example,

```
If Transaction.State='APPLIED' Then READ PROTECT Transaction FROM
ALL
```

When a business object is "read protected" the instances that match protection condition will not be read from the system. If an attribute is "read protected" its value will not be visible on any forms and cannot be changed.

4.2.15 `FIND` action

This action finds particular instances of a business object(s) in the system. There are several variations of the action:

1. `FIND ALL Id()`

where `Id()` indicates the name of the business object. The action written in this form will find all the existing instances of the specified business object. For example,

```
FIND ALL Account
```

2. `FIND (Id() | StringLiteral())`

where `Id()` or `StringLiteral()` indicate the name of the query to run. For example,

```
FIND 'Open accounts'
```

this action will run a query named 'Open accounts'. Note that the apostrophe must only be used if the query name contains space symbols otherwise the identifier of the query name can be used without the apostrophe.

3. `FIND Id() WHERE RuleCondition()`

where Id() indicates the name of a business object. The action written in this format will find all instances of the business object that match the specified condition. For example,

```
FIND Account WHERE Account.State = 'Open'
```

The following constructs can be optionally used after the `FIND` action (written in any of the above formats):

1. `ORDER BY AttributeIdentifier() (ASC | DESC)`

If this construct is used after the FIND action, the instances found by the action will be sorted by the specified attribute in the specified order. `AttributeIdentifier` indicates the attribute of the business object on which the sorting will be performed. The `ASC` keyword indicates the *ascending order* and the `DESC` keyword indicates the *descending* order. For example,

```
FIND ALL Account ORDER BY Account.Balance
```

finds all accounts sorted by their balances in the ascending order. If `ASC` and `DESC` keywords are omitted the `ASC` keyword is implied.

```
2. TAKE BEST (IntegerLiteral() | AttributeIdentifier())
```

If this construct is used after the FIND action, only the specified number of business object instances which match the specified criteria, will be found by the action. Typically this construct is used together with the ORDER BY keyword. For example,

```
FIND ALL Account ORDER BY Account.Balance DESC TAKE BEST 5
```

this action will find 5 accounts with the highest balance.

```
3. IN BATCHES OF IntegerLiteral()
```

If the action is likely to find many instances of the business object (hundreds or even thousands) their processing may take a while. In this case it is better to perform processing of instances in smaller chunks (batches). After processing of each batch is finished the results will be immediately committed to the system and stored in the database. If the batch size is not specified the system will only commit the results once all the instances have been processed. For example:

```
FIND Account WHERE Account.State = 'OPEN' AND Account.Balance >
1000 ORDER BY Account.Balance IN BATCHES OF 5
Perform some actions with found objects
```
(The actions will be performed in batches of 5 objects)

If `IN BATCHES OF` keyword is omitted the default batch size of 1000 is used. See also the "Batch Operations" section in the "Aware IM User Guide"

```
4. LIMIT ArithmeticExpression() "," ArithmeticExpression()
```

If this construct is used after the FIND action, only the specified rows will be returned. The first parameter indicates the first row number to be returned (starting from 1) and the second parameter indicates the number of rows to be returned. For example:

```
FIND Account WHERE Account.State = 'OPEN' AND Account.Balance >
1000 LIMIT 20,3
```

This will return 3 rows starting from row 20.

4.2.16 ENTER NEW action

This action is similar to the CREATE action in that it creates a new instance of a business object. Unlike the CREATE action it lets the user fill in the initial values of the business object. The action displays a form for the specified business object and waits for the user to provide the initial values of the attributes and submit them to the system. The syntax of the action is as follows:

```
ENTER NEW Id ()
```

where Id() is the name of the business object to create. For example,
`ENTER NEW Account`

Note: if the name of the business object group is specified in the action, ***Aware IM*** will first display a list of all members of the group and when the user selects the name of the required business object, ***Aware IM*** will display a form for this object.
Note: It is possible to optionally indicate initial values of the attributes using the WITH keyword. The syntax is the same as for the CREATE action. The form will be pre-populated with the supplied values (this initialization though is not available for groups – see previous note). For example:

```
ENTER NEW Account WITH Account.Name='John Smith'
```

Note: It is possible to optionally indicate the name of the specific form of the business object that will be used when entering the object, for example:

```
ENTER NEW Account USING 'Form for administrators'
```

where 'Form for administrators' is the name of the form that must be configured with the Account object.

Note: It is possible to optionally indicate the name of the specific section of the business object that will be displayed first, for example:

```
ENTER NEW Account USING 'Form for administrators' FORM_SECTION
'Personal details'
```

Note: By default the form of the object will be displayed in a pop-up window. If the process finishes immediately after the user enters form values, you can designate the form to be displayed in the current window or in a new tab. For example:

```
ENTER NEW Account AND VIEW
ENTER NEW Account AND VIEW IN TAB
```


4.2.17 `EDIT` action

This action displays a form of the existing instance of a business object to the user and waits for the user to provide the new values of the attributes. The syntax is as follows:

```
EDIT AttributeIdentifier()
```

where `AttributeIdentifier()` must be either the name of the business object or the name of its reference attribute.

For example, `EDIT Account` or `EDIT Account.Type`

Note: It is possible to optionally indicate the name of the specific form of the business object that will be used when editing the object, for example:

```
EDIT Account USING 'Form for administrators'
```

where '`Form for administrators`' is the name of the form that must be configured with the `Account` object.

Note: It is possible to optionally indicate the name of the specific section of the business object that will be displayed first, for example:

```
EDIT Account USING 'Form for administrators' FORM_SECTION
'Personal details'
```

4.2.18 `VIEW` action

This action is very similar to the `EDIT` action. The difference is that the process in which the action has been called does not wait for the user to change the values of the object and continues execution immediately after displaying the form. Also the `VIEW` action allows viewing the specific presentation of the business object. The syntax is:

```
VIEW AttributeIdentifier() [ USING Id() or StringLiteral() ]
[NOEDIT]
```

where `AttributeIdentifier()` must be the name of the business object.

For example, `VIEW Account`

`Id() or StringLiteral` optionally indicate presentation or form of the business object to be viewed, for example

```
View Accound USING 'Some fancy presentation'
```


where ‘Some fancy presentation’ is the name of the presentation that must be configured with the Account object.

Note: It is possible to optionally indicate the name of the specific section of the business object that will be displayed first, for example:

```
VIEW Account USING 'Form for administrators' FORM_SECTION
‘Personal details’
```

Note: If NOEDIT keyword is specified at the end of the action the form viewed will have all its input controls disabled, so that the user cannot change any of the values.

4.2.19 DISPLAY PERSPECTIVE action

This action displays the specified visual perspective on the screen.

```
DISPLAY PERSPECTIVE (Id() | StringLiteral())
```

where Id() or StringLiteral() is the name of the visual perspective to be displayed.

For example,
```
DISPLAY PERSPECTIVE ‘My perspective’
```

4.2.20 DISPLAY LAYOUT action

This action displays the contents of the specified tab of the main frame of the specified visual perspective on the screen.

```
DISPLAY LAYOUT (Id() | StringLiteral()) FROM_VP (Id() |
StringLiteral())
```

where the first pair of Id() or StringLiteral() is the name of the tab in the main frame to be displayed and the second pair is the name of the visual perspective.

For example,
```
DISPLAY Layout ‘My tab’ FROM_VP ‘My perspective’
```

4.2.21 DISPLAY DOCUMENT action

This action displays the specified document on the screen. The action has two variations:

1.
```
DISPLAY DOCUMENT AttributeIdentifier()[“USING StringLiteral()]
   [“MERGE INTO ONE”][“AS XLS”][“IN ZOHO”][“IN OO”][“NEW
   WINDOW”][MERGE WITH AttributeIdentifier()]
```


The action written in this format displays the contents of the attribute of the specified business object, provided that this attribute is of the `Document` type. Whatever document is stored in the attribute will be displayed – no document processing is performed. For example,

```
DISPLAY DOCUMENT Statement.StatementDocument
```

2. `DISPLAY DOCUMENT (Id() | StringLiteral()) [ "AS MESSAGE" ]["USING StringLiteral()] ["MERGE INTO ONE"] ["NEW WINDOW"][MERGE WITH AttributeIdentifier()]`

where `Id`() or `StringLiteral`() is the name of the document template to be displayed. Before the document template is displayed the action calculates and fills in attribute values used inside tags in the template (see also the "Document Generation" section in the "Aware IM User Guide").

Consider the following rules:

```
FIND Person WHERE Person.Name='John Smith'
DISPLAY DOCUMENT AccountOpeningLetter
```

where AccountOpeningLetter is the name of the document template. If this template has the following line:

```
Dear <<Person.Name>>,
```

it will be replaced with the line

```
Dear John Smith
```

when the template is displayed.

Note: If `AS MESSAGE` clause is specified the text of the document will be displayed in a pop-up message box. This option is only suitable for text or HTML document templates.

Note: You can specify a query in `USING` clause. Aware IM will use the query as the data source for the document. Aware IM will produce as many documents as there are instances returned by the query. If `MERGE INTO ONE` clause is not used Aware IM will display all instances on screen (the user will have to click on an icon next to each entry to see the document). If `MERGE INTO ONE` clause is specified Aware IM will automatically merge all documents into one document and show it to the user immediately.

Note: `AS XLS` clause can be used to display reports in the XLS format rather than the default PDF format

Note: `NEW WINDOW` clause can be used to display documents in a separate browser window

Note: `MERGE WITH` clause can be used to merge the displayed document with any other document stored in some object in an attribute of the DOCUMENT type (at the moment this is only supported for PDF documents), for example

```
DISPLAY DOCUMENT AccountOpeningLetter MERGE WITH
SystemSettings.TermsAndConditionsDoc
```

4.2.22 `DOWNLOAD DOCUMENT` action

This action is very similar to the `DISPLAY DOCUMENT` action, but instead of displaying the document on the screen, the action always downloads the document to the machine of the user. The syntax is:

```
1. DOWNLOAD DOCUMENT (AttributeIdentifier() | Id() |
   StringLiteral())
```

The action written in this format prints the contents of the attribute of the specified business object, provided that this attribute is of the `Document` type. Whatever document is stored in the attribute will be printed – no document processing is performed. For example,

```
PRINT DOCUMENT Statement.StatementDocument
```

4.2.23 `PRINT DOCUMENT` action

This action is very similar to the `DISPLAY DOCUMENT` action, but instead of displaying the document on the screen, it sends the document to the printer of the server. The action has two variations:

```
2. PRINT DOCUMENT AttributeIdentifier()
   [TO ArithmeticExpression()]
   [ <INTEGER> COPIES ]
```

The action written in this format prints the contents of the attribute of the specified business object, provided that this attribute is of the `Document` type. Whatever document is stored in the attribute will be printed – no document processing is performed. For example,

```
PRINT DOCUMENT Statement.StatementDocument
```

The TO clause indicates the name of the network server where the document will be sent for printing. If it is omitted the document will be sent to the default printer.

If number of copies is specified the specified number of documents will be printed, otherwise one copy will be printed.

Note: Files corresponding to known document types only will be printed.

Example:

```
PRINT DOCUMENT StaffReport 5 COPIES
PRINT DOCUMENT StaffReport TO `Network Printer 1` 5 COPIES
```

```
3. PRINT DOCUMENT (Id() | StringLiteral())
   [TO ArithmeticExpression()]
   [ <INTEGER> COPIES ]
```

where `Id`() or `StringLiteral`() is the name of the document template to be printed.

Before the document template is printed the action will calculate and fill in attribute values used inside tags in the template (see also the "Document Generation" section in the "Aware IM User Guide").

Consider the following rules:

```
FIND Person WHERE Person.Name='John Smith'
PRINT DOCUMENT AccountOpeningLetter
```

where AccountOpeningLetter is the name of the document template. If this template has the following line

```
Dear <<Person.Name>>,
```

it will be replaced with the line

```
Dear John Smith
```

when the template is printed.

4.2.24 `EXPORT DOCUMENT` action

This action exports the specified document into a file on disk. The action has two variations:

```
1. EXPORT DOCUMENT AttributeIdentifier()TO ("FILE" or
   "FOLDER")ArithmeticOperation()
```

The action written in this format exports the contents of the attribute of the specified business object, provided that this attribute is of the `Document` or `Picture` type. Whatever document is stored in the attribute will be exported – no document processing is performed. For example,

```
EXPORT DOCUMENT Statement.StatementDocument TO FOLDER
'c:/mydocuments'
```


2. `EXPORT DOCUMENT (Id() | StringLiteral())TO ("FILE" or "FOLDER")ArithmeticOperation()`

where `Id`() or `StringLiteral`() is the name of the document template to be exported. Before the document template is displayed the action will calculate and fill in attribute values used inside tags in the template (see also the "Document Generation" section in the "Aware IM User Guide"). Consider the following rules:

```
FIND Person WHERE Person.Name='John Smith'
EXPORT DOCUMENT AccountOpeningLetter TO FOLDER 'c:/mydocuments'
```

where AccountOpeningLetter is the name of the document template. If this template has the following line:

```
Dear <<Person.Name>>,
```

it will be replaced with the line

```
Dear John Smith
```

when the template is exported.

The `FOLDER` keyword specified in the arithmetic operation after the `TO` keyword identifies the directory in the file system where the file will be written. If the directory doesn't exist it will be created. The result of the arithmetic operation must be a string. The name of the file that the document will be written to will be the name of the original file that the document was created from. Note that if the file with this name already exists in the specified folder, it will not be overwritten. Instead another file with the variation of the original name (for example, originalName0) will be created.
If the `FILE` keyword is specified, then the arithmetic operation must contain the full path of the file that the document will be exported into. If a file with the specified name already exists it will be overwritten.

Note: `AS XLS` clause can be used to export reports in the XLS format rather than the default PDF format

4.2.25 `IMPORT DOCUMENT` action

This action imports the specified file on disk into the attribute of the specified object (provided that the attribute is of the `Document` or `Picture` type). The format of the action is:

```
IMPORT DOCUMENT AttributeIdentifier()FROM ArithmeticOperation()
```

For example,

```
IMPORT DOCUMENT Statement.StatementDocument FROM
'c:/mydocuments/myfile.doc'
```


The arithmetic operation specified after the `FROM` keyword identifies the directory in the file system where the file will be taken from. The result of the arithmetic operation must be a string.

4.2.26 `EXPORT` action

This action exports existing instances of the specified business object into a comma delimited text file (CSV file). The action has the following format:

```
EXPORT (Id() | StringLiteral()) TO ArithmeticOperation()
      [ "FOR UPDATE" ]
      [ "EXCLUDE RELATIONSHIPS" ]
      [ "EXCLUDE BINARY" ]
      [ "USING" ArithmeticExpression () ]
      [ "APPEND" ]
```

where `Id()` is the identifier of the business object to be exported, StringLiteral() is the name of the query to be exported and the `ArithmeticOperation()` is the expression identifying the name of the export file. If the file doesn't exist it will be created. The result of the arithmetic operation must be a string. For example:

```
EXPORT PersonalSavingAccount TO 'c:/mydocuments/psa.csv'
```

If the name of the business object is used, the instances available in the current context will be exported. If the name of the query is used ***Aware IM*** will run the query and export the results. If the query has attributes to display defined, only those attributes will be exported. Query names have to be enclosed in apostrophe. User defined queries are also supported.

There are several options that can be used with the action:

1. If the `FOR UPDATE` keyword is specified the ID attributes of the business objects will be written into the export file. If such a file is later imported into the system, the system will look for the objects with the specified ID's and modify them rather than insert the new ones. If the `FOR UPDATE` keyword is not specified the ID attributes of the objects will not be written into the export file and if such a file is later imported into the system, each record in the file will create the new instance of the business object. For example:

```
EXPORT PersonalSavingAccount TO 'c:/mydocuments/psa.csv' FOR
UPDATE
```

2. If `EXCLUDE RELATIONSHIPS` keyword is specified any relationships that the exported object has with other objects will not be written into the output file. By default all relationships are exported. For example:

```
EXPORT PersonalSavingAccount TO 'c:/mydocuments/psa.csv' EXCLUDE
RELATIONSHIPS
```


3. If `EXCLUDE BINARY` keyword is specified any binary attributes that the exported object has (such as `Pictures` and `Documents`) will not be written into the output file. By default all binary attributes are exported. For example:

```
EXPORT PersonalSavingAccount TO 'c:/mydocuments/psa.csv' EXCLUDE
BINARY
```

4. The USING expression allows you to use an existing export template. For example:

```
EXPORT PersonalSavingAccount TO 'c:/mydocuments/psa.csv' USING
`Account Template`
```

5. The APPEND clause allows you to append to an existing file rather than create a new one. For example:

```
EXPORT PersonalSavingAccount TO 'c:/mydocuments/psa.csv' APPEND
```

See also the “Export and Import” section in the “Aware IM User Guide”.

4.2.27 `IMPORT` action

This action imports the contents of the comma delimited text file (CSV) and creates or modifies instances of the specified business object. The action has the following format:

```
IMPORT Id() (FROM | FROM XML) ArithmeticOperation()
```

where `Id()` is the identifier of the business object to be imported and the `ArithmeticOperation()` is the expression identifying the name of the import file. The result of the arithmetic operation must be a string. For example:

```
IMPORT PersonalSavingAccount FROM 'c:/mydocuments/psa.csv'
```

If `FROM XML` clause is used the identifier indicates the XML string to import from, rather than the name of the import file.

The following options can be used with the action:

```
1. IN BATCHES OF IntegerLiteral()
```

If the action is likely to import many instances of the business object (hundreds or even thousands) their processing may take a while. In this case it is better to perform processing in smaller chunks (batches). After the processing of each batch finishes the results are immediately committed to the system and stored in the database. If the batch size is not specified the system will only commit the results once all instances have been processed. For example:


```
IMPORT PersonalSavingAccount FROM 'c:/mydocuments/psa.csv' IN
BATCHES OF 100
```

```
2. WITH VALIDATION
```

If this keyword is specified any rules associated with the business object being imported will be executed. If this option is omitted, the instances of the business object will be stored in the system but any rules associated with the business object will not be executed. If any rule fails for some record in the import file, this record will be ignored. For example:

```
IMPORT PersonalSavingAccount FROM 'c:/mydocuments/psa.csv' WITH
VALIDATION
```

Note: this option will slow down the import process considerably and is recommended when import files are not too large. On the other hand using this option will guarantee that only valid instances of the business object are stored in the system.

Note: First line of the input file must contain the description of the imported attributes. If this description contains the ID attributes, the system will look for the specified instances and modify them with the supplied data, otherwise new instances will be created.

```
3. KEEP RESULTS
```

If this keyword is specified imported objects will stay in the Context and will be available for further processing by rules. This option is not recommended if large number of objects is being imported.

```
4. CREATE IF NOT FOUND
```

If this keyword is specified the import file can be used both to update existing records and create new ones. If a record referred to in the file is not found ***Aware IM*** will create a new record.

See also the “Export and Import” section in the “Aware IM User Guide”.

```
5. USING
```

If this keyword is specified an import is performed using one of the user-defined import templates, for example:

```
IMPORT PersonalSavingAccount FROM 'c:/mydocuments/psa.csv' USING
'My import template'
```

4.2.28 `SET` action

This action allows setting the values of several attributes of a business object from a string. The format of the action is as follows:


```
SET [ATTRIBUTES OF] Id () FROM ArithmeticOperation() [ USING
StringLiteral]
```

where `Id()` is the identifier of the business object the attributes of which are being set and the `ArithmeticOperation()` is the expression identifying the string containing attribute names to set and their corresponding values.

The format of the import string must be one of the following:

1. `attrName1#attrValue1#attrName2#attrValue2` etc

The string represented in this format has attribute names followed by the corresponding values. The attribute names and values are separated by the "#" delimiter, for example

```
#Name#John Smith State#NEW#Balance#100.0
```

2. `attrName1 attrValue1 attrName2 attrValue2` etc

The string represented in this format also has attribute names followed by the corresponding values. The attribute names and values are separated by the space character (space character is thus not allowed inside attribute values, for example

```
Name John Smith State NEW Balance 100.0   - invalid
Name John State NEW Balance 100.0         - valid
```

3. The string has multiple lines where each line has a single attribute name and the corresponding value. The attribute name and value are separated either by the space character or "#" delimiter, for example:

```
Name John
State NEW
Balance 100
```

Instead of the space and "#" symbols you can use "USING" expression to specify the delimiter explicitly, for example:

```
SET Account FROM Name: 'John' USING ':'
```

The delimiter used here is the ":" symbol.

This action can be useful in a variety of situations, for example when processing the contents of the incoming e-mails (provided that they are formatted as described above).

4.2.29 UPDATE Action

The UPDATE action invokes evaluation of rules attached to the specified business object (provided that there are any). The format of the action is as follows:

```
UPDATE AttributeIdentifier()
```

4.2.30 `IMPORT RELATIONSHIPS` action

This action imports the contents of the comma delimited text file (CSV) that contains the relationships of the specified business object. The action has the following format:

```
IMPORT RELATIONSHIPS OF Id() FROM ArithmeticOperation()
```

where `Id()` is the identifier of the business object to be imported and the `ArithmeticOperation()` is the expression identifying the name of the import file. The result of the arithmetic operation must be a string. For example,

```
IMPORT RELATIONSHIPS OF PersonalSavingAccount FROM
'c:/mydocuments/psa.csv'
```

The following option can be used with this action:

```
IN BATCHES OF IntegerLiteral()
```

If the action is likely to import many instances of the business object (hundreds or even thousands) their processing may take a while. In this case it is better to perform processing in smaller chunks (batches). After the processing of each batch finishes the results are immediately committed to the system and stored in the database. If the batch size is not specified the system will only commit the results once all instances have been processed.
For example:

```
IMPORT PersonalSavingAccount TO 'c:\mydocuments\psa.csv' IN
BATCHES OF 100
```

If `IN BATCHES OF` keyword is omitted the default batch size of 1000 is used.

See also the “Export and Import” section in the “Aware IM User Guide”.

4.2.31 `DISPLAY MESSAGE` action

This action displays the specified information message to the user. The syntax is as follows:

```
DISPLAY MESSAGE [ASYNCH [DELAY INTEGER | CLOSABLE]
[TOP|TOP_LEFT|TOP_RIGHT|BOTTOM|BOTTOM_LEFT|BOTTOM_RIGHT]
ArithmeticOperation()[ MTITLE <STRING LITERAL> ]
```


where ArithmeticOperation is an operation that produces a string (usually just a string literal). For example,

```
DISPLAY MESSAGE 'Items have been created'
```

MTITLE key can be used to specify the title of the message box.

DISPLAY MESSAGE ASYNC displays temporary message in the specified corner of the screen. DELAY clause indicates how long the message will be visible. For example,
DISPLAY MESSAGE ASYNCH BOTTOM_LEFT 'Items have been created'

4.2.32 DISPLAY QUESTION action

This action displays a question to the user and waits for the reply. The string specified in the action identifying a question is displayed together with the Yes, No and Cancel buttons. The syntax of the action is as follows:

```
DISPLAY QUESTION ArithmeticOperation()[ MTITLE <STRING LITERAL> ]
```

where ArithmeticOperation is the operation that produces a string (usually just a string literal). For example,

```
DISPLAY QUESTION 'Do you want to print out a receipt? '
```

MTITLE key can be used to specify the title of the message box.

The reply of the user can be checked using the Reply attribute of the predefined object Question, for example:

```
If Question.Reply = 'Yes' Then …
```

4.2.33 DISPLAY URL action

This action goes to the specified URL. The syntax is as follows:

```
DISPLAY URL ArithmeticOperation() [ "IN NEW" | "IN MAIN" | "FULL
SCREEN"]
```

where ArithmeticOperation is an operation that produces a string (usually just a string literal). For example,

```
DISPLAY URL 'http://www.awareim.com'
```


"IN NEW WINDOW", "IN MAIN WINDOW", "IN CURRENT TAB", "NEW TAB" and "FULL SCREEN" options allow to display the URL in a new window, in the main window, in the current tab, in the new tab, or in the same window full screen respectively.

4.2.34 `PICK FROM` action

This action is similar to the `FIND` action in that it finds particular instances of a business object, but unlike the `FIND` action, it displays the instances found on the screen and waits for the user to pick a particular instance. The syntax of the action is almost exactly the same as that of the `FIND` action except that the `PICK FROM` keyword is used instead of `FIND`. For example,

```
PICK FROM Account WHERE Account.State = 'OPEN' ORDER BY
Account.Balance
```

Note:

1. `IN BATCHES OF` keyword that can be used in the `FIND` action is not applicable to the `PICK FROM` action.
2. By default the action will display a screen that allows selecting only one object. If multiple selection is required use the `PICK ONE OR MORE FROM` keyword instead of `PICK FROM`, for example

```
PICK ONE OR MORE FROM Account WHERE Account.State = 'OPEN' ORDER
BY Account.Balance
```

3. By default if a query finds just one instance of the object this instance will be displayed on the screen and the user will have to select it as usual. If you do not want the action to prompt for the single found instance, but use this instance straight away add `NO CONFIRMATION FOR ONE` keyword at the end of the action. For example,

```
PICK FROM Account WHERE Account.State = 'OPEN' ORDER BY
Account.Balance NO CONFIRMATION FOR ONE
```

4.2.35 `DISPLAY` action

This action is very similar to the `PICK FROM` action in that it displays the instances of the business object found by the action, however it does not expect the user to pick any particular instance. The syntax of the action is exactly the same as that of the `PICK FROM` action except that the `DISPLAY` keyword is used instead of `PICK FROM`. For example,

```
DISPLAY Account WHERE Account.State = 'OPEN' ORDER BY
Account.Balance
```


4.2.36 `PRINT FORM` action

This action prints the specified form of the specified business object on the client's screen. Note that unlike the `PRINT DOCUMENT` action that prints the document on the server, the `PRINT FORM` action prints the form on the *clients's* machine inside a browser. For example,

```
PRINT FORM Main OF Account
```

This action prints the form with the name "Main" of the `Account` object.

4.2.37 `EXECUTE PROGRAM` action

This action allows execution of an external program.

The syntax is:
`EXECUTE PROGRAM ArithmeticExpression() [ "NO WAIT" ]`

Where `ArithmeticExpression` identifies the path to the external program. The `NO WAIT` clause indicates that the system should not wait for the external program to complete.

4.2.38 `EXEC_SP` action

This action allows execution of a database stored procedure.

The syntax is:

```
"EXEC_SP" SPName
[ "OF" DatabaseEnvironment ]
[
  "WITH" ParamName "=" ArithmeticExpression ()[ "INOUT")|"OUT" ]
  (
      ","
      ParamName "=" ArithmeticExpression ()[ "INOUT")|"OUT" ]
   )*
]
[
      "RETURN" ObjectName
]
```

where:

- `SPName` is the name of the stored procedure to execute (must be enclosed in apostrophe)
- `DatabaseEnvironment` if present indicates the name of the database environment where the stored procedure is defined (see objects persisted in external databases in the User Guide). If not present the native Aware IM database is used.
- `ParamName` is the name of the parameter of the stored procedure (if stored procedure requires a parameter. A parameter can be "IN", "OUT" or "INOUT". If parameter type is omitted "IN" is assumed
- `ArithmeticExpression` an Aware IM expression to initialize input parameters with. If the parameter is of OUT or INOUT type this expression must be the name of an attribute of a business object

- `ObjectName` if the stored procedure returns a result set (the result of the SQL SELECT statement) Aware IM can convert each record to the instance of the specified object. An object that is persisted in the database table of the SELECT statement must be defined. The resulting object instances are either placed in the Context if the EXEC_SP action is used in rules or returned by a query if the action is used in a rule form of the query

Examples:

```
EXEC_SP 'procAlertGetAll' RETURN Alert
```
This stored procedure does not require any parameters. It "selects" all alert records in the native database, that Aware IM will automatically convert to instances of the Alert object and put in the context or make available for a query

```
EXEC_SP 'procAlertGet' WITH '@alerID'=1 RETURN Alert
```
This stored procedure returns alert record with id=1

```
EXEC_SP 'procAlertGet' WITH '@alerID'=SPParam.AlertId RETURN
Alert
```
This stored procedure returns alert record with id taken from the `AlertId` attribute of the `SPParam` object

```
EXEC_SP 'procAlertGetOut' WITH
'@alerID'=SPParam.AlertId,'@alertName'=SPParam.AlertName OUT
```
This stored procedure returns alert record with id taken from the `AlertId` attribute of the `SPParam` object. The name of the alert is then written into the `AlertName` attribute.

```
EXEC_SP 'proc1' OF SQLServer WITH '@param1'=1,'@param2'=2 RETURN
SomeObject
```
This stored procedure returns the specified records from an external database identified by the name `SQLServer`.

4.2.39 CONNECT TO EMAIL action

Using this action allows you to handle incoming e-mails dynamically, from a process. This action starts listening to the specified incoming e-mail account. The syntax is:

```
CONNECT TO EMAIL AttributeIdentifier()
```

The attribute identifier specified in the action identifies the e-mail account to connect to. It is expected that this identifier points to the object containing the following mandatory attributes:

- `MailHost` – email host server
- `MailUser` – user name of the e-mail account on this server
- `MailPassword` – password of the e-mail account on this server
- `AccountName` – a unique name identifying this connection (which is then used in the `DISCONNECT FROM EMAIL` action – see below).

The following attributes are optional. If not provided default values are used:

- `MailPort` (Number, default value is 110) – port which the e-mail server uses
- `MailCheck` (Number, default value is 300) – how frequently to check for new emails (in seconds)
- `MailSSL` (Yes/No, default is No) – whether e-mail account requires SSL
- `MailTLS` (Yes/No, default is No) – whether e-mail account requires TLS

An example of the action:

```
CONNECT TO EMAIL EmailAccount
```

This assumes that the application contains the EmailAccount object with the attributes above. At runtime the values of this object identify the e-mail account to connect to.

4.2.40 DISCONNECT FROM EMAIL action

This action stops handling incoming e-mails started by the `CONNECT TO EMAIL` action. The syntax is:

```
DISCONNECT FROM EMAIL AttributeIdentifier()
```

Just like the CONNECT TO EMAIL action the AttributeIdentifier() specifies the object with the attributes identifying the e-mail account.

4.2.41 SAVE SCREEN action

This action allows you to save image of a chart displayed on the current screen or the entire displayed screen in an attribute of the Picture or Document types (in the latter case the image is saved as PDF file). The syntax is:

```
SAVE SCREEN [StringLiteral()] TO AttributeIdentifier() [ AS PDF ]
```

For example:

```
SAVE SCREEN 'my chart id' TO Object.PictureAttribute
```

The action above saves the chart identified by id 'my chart id' in the picture attribute of some object in the Context. The id of the chart can be specified when defining a chart query.

```
SAVE SCREEN TO Object.PictureAttribute
```

The action above saves the current screen in the picture attribute of some object in the Context.

```
SAVE SCREEN TO Object.DocumentAttribute AS PDF
```


The action above saves the current screen in the document attribute of some object in the Context as PDF file.

4.2.42 RENAME DOCUMENT action

This action allows you to rename the document stored in an attribute of the Document type. The syntax is:

```
RENAME DOCUMENT AttributeIdentifier() TO ArithmeticExpression()

For example: RENAME DOCUMENT Object.DocAttribute TO `abc.doc`
```

4.2.43 LOGOUT action

This action forces logout of the specified user.

The syntax is:

```
"LOGOUT" AttributeIdentifier()
```

where AttributeIdentifier is the user to logout, for example:

```
FIND SystemUser WHERE SystemUser.LoginName='john'
LOGOUT SystemUser
```

4.2.44 LOGOUT2 action

This action forces logout of all users with the same login name as the currently logged in user. But it does not logout currently logged in user. This can be useful if you want to forcefully logout all other users who are logged in already with the same name as current user.

The syntax is:

```
"LOGOUT2"
```

4.2.45 CLOSE TAB action

This action forces closes all tabs with the specified id. Id's can be assigned when defining an operation or a menu item that directs its output to a new tab (settings of the "New Tab" output).

The syntax is:

```
"CLOSE TAB" STRING LITERAL
```

where `STRING LITERAL` is the id of the tabs to close, for example:

```
CLOSE TAB 'my tab'
```


4.2.46 `EXEC_STRING` action

This action executes the specified string as if it was an action of the Rule Language. This allows configurators to define the rule action dynamically.

The syntax is:

```
"EXEC_STRING" ArithmeticExpression()
```

where `ArithmeticExpressio` defines a string to be executed, for example:

```
Object.Attribute = 'FIND ALL Customer'
EXEC_STRING Object.Attribute
```

4.2.1 `EXEC_SQL` action

This action executes the specified SQL string by invoking the database engine.
The syntax is:

```
EXEC_SQL ArithmeticExpression ()[ "OF" <IDENTIFIER>)] [ RETURN Id() ]
```

where `ArithmeticExpressio` defines a string to be executed, for example:

```
EXEC_SQL `UPDATE CRM_Customer SET Name='John'`
```

If SQL to be executed performs a SELECT specify the object name that the SQL returns, for example:

```
EXEC_SQL `SELECT CRM_Customer WHERE FirstName='Jane'` RETURN
Customer
```

You have to be careful when using this acton as you can stuff up your database if you do something wrong. This action is only recommended for advanced users who understand how Aware IM manages the database.

The identifier after "OF" can be used to identify a particular "database environment", defined when connecting Aware IM to external databases. If not present the native Aware IM database is used

4.2.2 `LOG2` action

This action puts the specified string into the Exection Log, so that it becomes visible in the Log Viewer – see the User Guide for more details about the Execution Log and Log Viewer.

The syntax is:

```
"LOG2" ArithmeticExpression()
```

where `ArithmeticExpressio` defines a string to be placed in the log, for example:


```
LOG2 'My string'
```

4.2.3 LOG2 CONTEXT action

This action dumps the current Context into the Exection Log, so that it becomes visible in the Log Viewer – see the User Guide for more details about the Context, Execution Log and Log Viewer.

The syntax is:

```
"LOG2 CONTEXT"
```

4.2.4 COMMIT TRANSACTION action

This action forcefully commits the current database transaction. The next action of the process (if any) starts a new transaction. See "Rules and Transactions" section in the User Guide for more details.

The syntax is:

```
"COMMIT TRANSACTION"
```

4.2.5 MOBILE CAMERA SNAP INTO action

This action opens a camera on a mobile device that the user logged in from, allows the user to take a photo and then writes the result into the specified Picture attribute of the specified business object. Note that this action can only be used in native mobile applications – it will not work in a browser.

The syntax is:

```
"MOBILE CAMERA SNAP INTO " AttributeIdentifier()
[
  "WITH" StringLiteral() "=" StringLiteral()
  ("," StringLiteral() "=" StringLiteral())*
]
```

where `AttributeIdentifier` is the identifier of the attribute to write the result into (the attribute must be of the Picture Type), for example:

```
MOBILE CAMERA SNAP INTO MyObject.MyPictureAttribute
```

You can also optionally specify different properties defining how the camera will behave. The following parameters can be used (see more details here: https://github.com/apache/cordova-plugin-camera)

Name	Type	Default	Description
quality	number	50	Quality of the saved image, expressed as a range of 0-100, wher resolution with no loss from file compression. (Note that inform camera's resolution is unavailable.)
allowEdit	Boolean	true	Allow simple editing of image before selection.
encodingType	EncodingType	JPEG	Choose the returned image file's encoding.
targetWidth	number		Width in pixels to scale image. Must be used withtargetHeight. constant.
targetHeight	number		Height in pixels to scale image. Must be used withtargetWidth. constant.
correctOrientation	Boolean		Rotate the image to correct for the orientation of the device duri
saveToPhotoAlbum	Boolean		Save the image to the photo album on the device after capture.
popoverOptions	CameraPopoverOptions		iOS-only options that specify popover location in iPad.
cameraDirection	Direction	BACK	Choose the camera to use (front- or back-facing).

For example:

```
MOBILE CAMERA SNAP INTO MyObject.MyPictureAttribute WITH
'targetWidth'='100','targetHeight'='100','cameraDirection'='1','allowEdit'='fa
lse'
```

4.2.6 `MOBILE CAMERA GET INTO` action

This action opens a photo library on a mobile device that the user logged in from, allows the user to select a picture from the library and then writes the result into the specified Picture attribute of the specified business object. Note that this action can only be used in native mobile applications – it will not work in a browser.

The syntax is:

```
"MOBILE CAMERA GET INTO " AttributeIdentifier()
[
  "WITH" StringLiteral() "=" StringLiteral()
  ("," StringLiteral() "=" StringLiteral())*
]
```

where `AttributeIdentifier` is the identifier of the attribute to write the result into (the attribute must be of the Picture Type), for example:

```
MOBILE CAMERA GET INTO MyObject.MyPictureAttribute
```

You can also optionally specify different properties defining how the photo library will behave. See the `MOBILE CAMERA SNAP INTO` action for more details.

4.2.7 `MOBILE GET LOCATION INTO` action

This action reads the current location on a mobile device, that the user logged in from, and writes the results into the attributes of the specified `MGeoLocation` object. For example,

```
MOBILE GET LOCATION INTO MGeoLocation
```

The results will be written into the specified instance of the `MGeoLocation` object (longitude, latitude, altitude etc).

4.2.8 `MOBILE START LOCATION WATCH INTO` action

This action starts tracking changes to the current location of the user on a mobile device that he used to log in from. The changes are written into the specified instance of the `MGeoLocation` object. For example:

```
MOBILE START LOCATION WATCH INTO MGeoLocation
```

4.2.9 `MOBILE STOP LOCATION WATCH FROM` action

This action stops tracking changes to the current location of the user on a mobile device that he used to log in from. For example:

```
MOBILE STOP LOCATION WATCH FROM MGeoLocation
```

4.2.10 `MOBILE SUBSCRIBE` action

This action subscribes the mobile device of the user that he logged in from to receive push notifications from the system. This action only works in native mobile applications – it doesn't

work in the browser.Notifications are sent by using the `MOBILE PUSH` command (see below). As the result of the successful subscription Aware IM automatically creates an instance of the `MobilePhone` object and puts it into the Context. The instance can then be used to send notifications to. For example:

```
MOBILE SUBSCRIBE
LoggedInRegularUser.Phone = MobilePhome
```

See the How To document for more details about push notifications.

4.2.11 `MOBILE PUSH` action

This action sends the specified push notification to the mobile phones (`MobilePhone` objects) that are currently in the Context of the process. This action only works in native mobile applications – it doesn't work in the browser. For example,

```
FIND ALL MobilePhone
CREATE MobilePushNotification WITH
MobilePushNotification.Title='Test',MobilePushNotification,Subject='This is my
test notification',MobilePushNotification.Sound='default'
MOBILE PUSH MobilePushNotification TO MobilePhone
```

Please see the How To document for more details about push notifications.

4.2.12 `EXEC_SCRIPT` action

This action runs the specified Javascript inside the browser of the current user that started the process with this action. The syntax is:

```
"EXEC_SCRIPT" ArithmeticExpression()
```

where `ArithmeticExpression` is either a string or an attribute containing a string with a script.

For example:

```
EXEC_SCRIPT ‘AwareApp.closeComponent(parser.m_widgetInfo,false,false);‘
```

Aware IM exposes a special object called "parser" to the script run through this action. This object represents a form or query parser that the process has been started with. This way the developer can use this object to perform some operations with a form or a query before or after the process starts. The above script, for example, will close the current form or query if the process that contains this action has been started from the form or query. If the process hasn't been started from the form or query the "parser" object is null.

Some more example of the script execution. The following one will select the second tab in a visual perspective:

```
EXEC_SCRIPT ‘AwareApp.getMainTabPanel().select(1);‘
```


The following script select a tab in a visual perspective with id “User”:

```
EXEC_SCRIPT `var tabIdx = $('#' + AwareApp.m_mainTabPanelId).find('[aw-tab-
id=User]').parent().index()-1; AwareApp.getMainTabPanel().select(tabIdx);`
```

Please see the Programmers Reference document for more details on how to use Javascript in Aware IM.

4.3 Process call action

This type of actions makes it possible to run a process. The syntax of the action just includes the name of the process to run. For example,

```
If Policy.State = 'Open' Then CalculatePremium
```

In this rule `CalculatePremium` is the name of the process to run. (`RUN PROCESS` keyword can be optionally specified as well, for example `RUN PROCESS CalculatePremium`)

If a process requires process input then the instances of the business objects representing the process input will be automatically taken from the current Context when the action is executed (see also the “Context of Rule Execution” section in the “Aware IM User Guide”). It is also possible to explicitly indicate this input using the following syntax:

```
USING (AttributeIdentifier())+
```

For example,

```
If Policy.State = 'Open' Then CalculatePremium USING Policy
```

The process input must be used explicitly in the action if it represents a reference attribute. For example,

```
If Customer.CurrentPolicy.State = 'Open' Then CalculatePremium
USING Customer.CurrentPolicy
```

It is possible to indicate that the process should be run asynchronously – that is, if a system is executing an ordered collection of rules, one of which calls the process asynchronously, the next rule will execute immediately without waiting for the process to finish. To indicate that the process should run asynchronously, add `NOWAIT` keyword after the process name. For example,

```
CalculatePremium NOWAIT
```


5 Special usage of rules

There are several places in the software where the special format of the rule expressions is used.

5.1 Rule tables

Inside rule tables some rule expressions can be represented in the simplified format. This can happen only if an attribute identifier is specified in the header of the rule table column or row. For example, if `Account.Balance` attribute is in the header of the table column and the value of the cell is 1000, the value is the short form of the following expression:

`Account.Balance` = 1000.

The following expressions can be shortened in cells of a rule table:

5.1.1 Relational expression

Compared to the full form of the relational expression (see section 3.2) the shortened version used in rule tables has attribute identifier omitted from the expression. Also when "=" is used it is also omitted from the expression. Examples of shortened relational expressions and the corresponding full versions follow (these assume that the header of the rule table column is `Account.Balance`):

```
1. 1000
```

the corresponding expression in the full form is:

```
Account.Balance = 1000
```

```
2. <1000
```

the corresponding expression in the full form is:

```
Account.Balance < 1000
```

```
3. >= Transaction.Amount – 500
```

the corresponding expression in the full form is:

```
Account.Balance >= Transaction.Amount – 500
```


5.1.2 String Expression

The short form of the `String Expression` (see section 3.3) has attribute identifier omitted from the expression. The examples, which assume that `Account.Name` is the header of the column, follow:

```
1. CONTAINS 'ith'
```

The corresponding expression in the full form is

```
Account.Name CONTAINS 'ith'
```

```
2. STARTSWITH 'John'
```

The corresponding expression in the full form is

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

5.1.3 List Expression

The short form of the List Expression (see section 3.5) has the attribute identifier omitted from the expression. Also OR is used instead of comma to separate members of the list. The examples, which assume that `Account.State` is the header of the column, follow:

```
1. 'OPEN' OR 'CLOSED'
```

the corresponding expression in the full form is:

```
 Account.State IN 'OPEN', 'CLOSED'
```

```
2. 'OPEN' OR 'CLOSED' OR 'SUSPENDED'
```

the corresponding expression in the full form is

```
Account.State IN 'OPEN', 'CLOSED', 'SUSPENDED'
```

5.1.4 Range Expression

The short form of the `Range Expression` (see section 3.6) has the attribute identifier omitted from the expression. Also AND is always used to separate boundaries of the range. There is also an alternative form, which has two boundaries separated by "-" sign. The examples, which assume that `Account.Balance` is the header of the column, follow:

```
1. BETWEEN 1000 AND 5000
```

The corresponding expression in the full form is:


```
Account.Balance BETWEEN 1000 AND 5000
```

```
2. 1000-5000
```

the corresponding expression in the full form is:

```
Account.Balance BETWEEN 1000 AND 5000
```

5.1.5 `WAS CHANGED` expression

The short form of the `WAS CHANGED` expression (see section 3.7) has the attribute identifier omitted from the expression. The examples, which assume that `Account.State` is the header of the column, follow:

```
1. WAS CHANGED
```

The corresponding expression in the full form is:

```
Account.State WAS CHANGED
```

```
2. WAS CHANGED TO 'CLOSED'
```

The corresponding expression in the full form is:

```
Account.State WAS CHANGED TO 'CLOSED'
```

5.1.6 `IS UNDEFINED` expression

The short form of the `IS UNDEFINED` expression (see section 3.8) has the attribute identifier omitted from the expression. The Examples, which assume that `Account.Name` is the header of the column, follow:

```
1. IS UNDEFINED
```

The corresponding expression in the full form is:

```
Account.Name IS UNDEFINED
```

```
2. IS DEFINED
```

The corresponding expression in the full form is:

```
Account.Name IS DEFINED
```


5.1.7 Assignment action

The short form of the assignment action (see section 4.1.1) has the attribute identifier omitted from the action. The examples, which assume that `Account.Balance` is the header of the column, follow:

```
1. 1000
```

the corresponding action in the full form is:

```
Account.Balance = 1000
```

```
2. Transaction.Amount*100
```

The corresponding action in the full form is:

```
Account.Balance = Transaction.Amount * 100
```

5.1.8 Incremental action

The short form of the incremental action (see section 4.1.2) has the attribute identifier omitted from the action. “+” and “-“ signs can also be used to indicate whether the `INCREASE` or `REDUCE` operation is used. The examples, which assume that `Account.Balance` is the header of the column, follow:

```
1. INCREASE BY 1000
```

The corresponding action in the full form is:

```
INCREASE Account.Balance BY 1000
```

```
2. +1000
```

the corresponding action in the full form is:

```
INCREASE Account.Balance BY 1000
```

```
3. -20%
```

the corresponding action in the full form is:

```
REDUCE Account.Balance BY 20%
```


5.2 Usage of rules inside tags

Rule expressions can be used inside special tags in documents. The start of the tag is denoted by the symbol "<<" and the end of the tag – by the symbol ">>". The following expressions can be used inside tags:

1. `Arithmetic Operations` (see section 3.1)
2. Conditional expressions
3. Special expressions identifying boundaries of document sections (MS Word documents only).

5.2.1 Arithmetic Operations

Any `Arithmetic Operation` can be used inside a tag, for example:

`<<Account.Balance>>` or `<<Account.Balance + 1000>>`.

`Arithmetic Operation` inside a tag can be optionally followed by the `Formatting Expression`, which indicates how the value of the tag will be presented to the user when the tag contents is calculated at run time. The `Formatting Expression` is separated from the `Arithmetic Operation` by comma or @ symbol. If no formatting expression is specified the value will be formatted according to its default format, which is obtained from the configured presentation information of the attribute(s) participating in the `Arithmetic Operation` inside the tag.

There are four types of the formatting expression:

1. The Date/Time format
2. The Number format
3. The Duration format.
4. HTML format

The examples of rule expressions using formatting expressions inside tags follow:

```
1. <<Account.Balance, 00000>>
2. <<Account.OpeningDate, dd/MM/yyyy>>
3. <<ActivityRecord.TimeSpent@WWw DDd HH:mm>>
4. <<Account.Description,html>> (Description attribute is assumed
   to store text in the HTML format)
```

5.2.2 Conditional Expressions

It is possible to indicate that attribute values or static text should be included in a document conditionally. This can be done using conditional expression. The format of this expression is that of any rule that uses condition, except that this expression must use special action called "`SHOW`". For example,


```
1. <<IF Account.Balance > 1000 Then SHOW Account.Holder.Name>>
2. <<IF Account.Balance > 1000 Then SHOW 'Account balance is
   greater than 1000'
```

5.2.3 Expressions marking section boundaries

It is possible to conditionally include entire sections of a document template in the final document. To do this you need to specify the condition of inclusion, mark the start of the section to be included in the document and mark the end of the section in the document. Condition is specified using the conditional expression with the special action "`SHOW SECTION_START`". This also marks the start of the section. To mark the end of the section include, the following tag: `<<SECTION_END>>`. For example,

```
<<IF Account.Balance > 1000 THEN SHOW SECTION_START>>
The text of the section follows …
<<SECTION_END>
The rest of the text follows
```

Note: conditional inclusion of document sections is only available for MS Word document templates. MS Word document templates also support inclusion of sub-documents.

5.3 Report Designer

When rule expressions are used inside tags in the Report Designer the following additional construct can be used:

```
{ Id () }
```

where `Id()` indicates the name of the report parameter. Before the report using some parameter(s) is executed the user is asked to provide the specific values of the report parameter(s). During the report calculation report parameters are replaced with values provided by the user. For example:

```
<<{REPORT_NAME}>>
```

Note: the report parameter expression can only be used inside the Report Designer.

5.4 Queries

In ***Aware IM*** queries can be defined using the Rule Language. The only Rule Language construct that can be used when constructing queries is the `FIND` action (see section 4.2.14).

The action can use "?" symbol to indicate that the value of the attribute used in the query should be prompted to the user when the query is executed. For example,


```
FIND Account WHERE Account.Balance > ?Balance
```

When such a query is executed, the user is asked to enter the specific value of the account balance. The "Balance" string after the question mark indicates the name of the prompt that will be displayed to the user. See also the "Configuring Queries" section in the "Aware IM User Guide".

6 Notes on rule execution

6.1 Type conversions

When performing arithmetic calculations with attribute values and numeric constants ***Aware IM*** automatically performs the necessary type conversions if arguments are of different types. The following conversion rules apply:

- Addition of two strings causes the strings to be concatenated, for example `'John' + ' ' + 'Smith'` produces `'John Smith'`
- Addition of a string and a number produces a string, for example `'Number' + Account.Balance` produces `'Number1000'` if the value of `Account.Balance` is 1000
- If a value is undefined it is omitted from calculations, for example if the value of `Account.Balance` is 1000 and the value of `Transaction.Amount` is undefined, the value of `Account.Balance – Transaction.Amount` will be 1000.
- Strings may not participate in any arithmetic calculations other than addition.
- References may not participate in arithmetic operations.
- A number added (subtracted) to (from) a date produces the date which is "value" days greater or less than the original date, where "value" is the value of the number. For example, `Account.OpeningDate + 7` produces a date that is a week after the account opening date.
- A number added (subtracted) to (from) a timestamp produces the timestamp which is "value" hours greater or less than the original timestamp, where "value" is the value of the number. For example, `Transaction.Timestamp – 3` produces the timestamp that is 3 hours before the transaction timestamp.
- Duration added or subtracted from a date produces date.
- No other arithmetic operations with date/timestamp are allowed.

Note: no arithmetic operations with date/timestamp may be used inside the `FIND` action.

- Duration added to or subtracted from another Duration produces Duration
- Attributes of the Yes/No type may not participate in arithmetic calculations.
- Attributes of the `Document` type may not participate in arithmetic calculations, although it is possible to assign a value to the attribute of the `Document` type. For example, `Statement.Doc = 'StatementTemplate'` – assigns the specified template to the attribute of the business object. This will also resolve contents of tags if the template has them.

Another example, `Statement.Doc = OtherObject.Doc` – assigns the document of `Statement` business object to be the exact copy of the document stored in `OtherObject`.

6.2 Comparison

The following rules apply when comparing values in the `Relational Expression`:
All attribute types can be compared for equality or non-equality (even references), provided that the types on both sides are compatible. If the types are the same on both sides, they are always compatible. If they are different, the following rules apply:

- If strings are compared with numbers, numbers are converted to strings, for example, the following will only be true if the value of `Account.Name` is '1000': `Account.Name = 1000`
- If a value of an attribute is undefined it is considered to be 0 for comparison purposes.

Only numbers, dates and durations can be compared using other operators ("<", ">", "<=", ">=").

6.3 References

It is possible to use references and reference lists inside rule conditions and assign reference and reference lists in actions. For example, the following rule:

```
If Customer.Policies.Cartype='Wagon' Then
Customer.Policies.Premium=100
```

will check every policy in the list of policies of a customer and if a car type of the policy is wagon will set the premium of such policy to 100.

The following rule:

```
If Policy.CarType='Wagon' Then Customer.CurrentPolicy=Policy
```

will set the policy, the car type of which is "wagon", as the policy of the customer.

If multiple references are used in a rule condition the corresponding action will be triggered if the condition holds for ANY value of the reference in the reference list. For example,

```
If Customer.Policies.CarType='Wagon' Then …
```

The action in this rule will be executed if there is any `Policy` that the `Customer` owns that has the Wagon car type.

