

2. COMDALE/X Terminology And Concepts

2.1 Overview

The principal components of a COMDALE/X expert system are the:

- Knowledge Base;
- Inference Engine;
- User Interface;
- Utilities.

The COMDALE/X Expert System Tool is comprised of the **inference engine, user interface** and **utilities**. Configuring a knowledge base is required for each expert system application.

All the information that relates to the knowledge that humans use to make decisions in a particular domain is held in the **Knowledge Base**. The knowledge base therefore contains objects, classes, rules and procedures.

The **Inference Engine** processes or manipulates that knowledge to make decisions.

The **User Interface** provides the window for one to observe and communicate with COMDALE/X during the development and use of an expert system application.

COMDALE/X **Utilities** facilitate various approaches to representing and displaying information as well as providing for security of the knowledge base.

The COMDALE/X tool can be used to develop stand alone or embedded expert system applications.

For application development you can use graphical editors or your own ASCII editor. For delivery you can customize our interface, provide your own, or embed our technology using our Source Code Developers Library.

2.2 Knowledge Base

The knowledge base is an assembly of the knowledge that a human might use to conceptualize information and make decisions in a particular domain.

The knowledge base is comprised of four elements:

- Objects;
- Classes;
- Rules;
- Procedures.

Objects represent factual information.

The structural relationships of the facts in a hierarchical classification are termed **classes**.

Rules are the complex relationships which are formulated between the facts.

Procedures are used to dictate the application of rules and the manipulation of classes and objects.

Knowledge base contains Objects, Classes, Rules and Procedures.

These elements along with various attributes, which describe them, are called **knowledge base units**. Knowledge base files have the extension **.knw**.

COMDALE/X compiles a knowledge base when it is loaded for use. Maintaining and consulting a knowledge base requires loading the system only once. Objects, classes, rules etc. are incrementally compiled as they are modified with the graphical editors.

A keyword triplet is comprised of an object, attribute and value.

2.2.1 Objects

Objects are physical or conceptual entities in the real world.

Each object, which is created by the developer of the knowledge base, has one or more **attributes, which** describe a characteristic of that object.

The combination of an object, attribute and value (which describe the state of an object's attribute) is used to define a variable. These variables are called **keyword triplets**, and have associated with them a list of facets e.g. customized questions and fuzzy sets.

Example

'tank150'.

tank 150 may have a height or a level; and these (height or level) are considered attributes. A description of the entity is completed with a **value** attached to it. Values explain the particular or special nature of attributes. For example, level of tank 150 can be low – low, being the value.

Example

'pump12'

This object may have attributes such as 'speed' and 'flowrate'. These attributes could have been associated with this object by inheriting them from a class to which the object belongs, or by definition in a rule, procedure or by use of the object editor. The object – attributes, 'pump12.speed', 'pump12.flowrate' may have values 'high', 'low', etc. In these cases keyword triplets would be defined as 'pump12.speed.high' and 'pump12.flowrate.low'. Each object can have an unlimited number of attributes and each attribute can have an unlimited number of values.

The combination of object, attribute and value is termed a **keyword triplet**.

In COMDALE/X, a keyword triplet is written in the form:

object.attribute.value

Example:

tank150.level.low

The object, attribute and value parts of a keyword triplet are called **tokens** and are separated by a period. Keyword triplets can be of different types, according to the kinds of value they may have:

logical (or Boolean)	e.g. tank.level.low
string	e.g. tank.manufacture.@string
numerical	e.g. tank.level_setpoint.@float
integer	e.g. tank.number.@integer
date	e.g. tank.delivery.@date
time	e.g. tank.fill_time.@time

Object Any name consisting of between 1-39 alphanumeric characters, cannot start with a number. Spaces cannot exist, underscores are permitted.

Attribute Any name consisting of between 1-39 alphanumeric characters, cannot start with a number. Spaces cannot exist, underscores are permitted.

Value Any name consisting of between 1-39 alphanumeric characters, cannot start with a number. Spaces cannot exist, underscores are permitted.

Associated with each keyword triplet is a set of **facets**, for example, the facet of uncertainty and time. **Uncertainty** is used to describe, for instance, how sure we are that the level of tank 150 is low; or time being used to describe when the level of tank150 became low. Other facets include **fuzzy sets, customized questions, restrictions on values, sources of information**, etc. These facets are configured during development of the knowledge base.

Inheritance

Inheritance of attributes occurs when an object is assigned to be a member of a class. It inherits all the attributes of the class to which it belongs as well as those of the superclasses of this class. (if one is defined)

Creating Objects

Each object has a unique name and a set of attributes. Within COMDALE/X the object editor can be used to define objects, however, this is not always necessary.

Rather than declaring each object using the object editor, COMDALE/X allows you to use objects in rules, procedures and facets. When these are compiled (by clicking on the appropriate **OK** button in the editors), the objects are created automatically. You may then view them or edit their facets using the object editor. You only need to specify membership of objects in classes in the class or object editors.

“Object Editor” Window

The “Object Editor” is used to create, maintain, and delete objects and their corresponding attributes and facets. It can be found under the **Editor** button in the main bar menu.

Dynamic and Interpretation objects are created during runtime.

2.2.2 Runtime Objects

During a consultation session you can redefine keyword triplets used in rules. Two mechanisms are provided for ultimate flexibility in this definition, they are:

- interpretation objects
- dynamic objects

Interpretation

objects are concatenated.

2.2.2.1 Interpretation Objects

In developing the knowledge base you can use interpretation objects to create new objects during the inference process by linking string.

Syntax:

`\string\<embedded triplet variable>`

The pair of angle brackets (`<>`) is used to enclose a keyword triplet of type numerical, string, date, or time, while the slashes (`\ \`) encloses a string which is to be concatenated with the value of the triplet variable to create a new object during runtime.

Example: The string “tank” and the value of a keyword triplet **tank.number.@integer = 10** (an integer type keyword triplet) can be combined to produce the object tank 10. This is done in a statement with the following representation:

`IF\tank\< tank.number.@integer>.level.high`

Based on a value of **tank.number.@integer = 10** this expression will be interpreted to be:

`IF tank10.level.high`

Example: `if \tank\< tank.number.@i > .level.@f < 100`

In this condition statement we have an embedded triplet variable **tank.number.@i**. assume that **tank.number.@i** has been previously instantiated to a value of 10. This implies that the condition above is in reality,

`If tank10.level.@f < 100`

Notes:

1. In this example the embedded keyword triplet can have different integer values depending on how it was instantiated.
2. The embedded information can also be a function call.
3. The embedded information can also be a constant.
4. The string enclosed in the slashes (`\ \`) is not a string triplet variable.

Dynamic objects are sets of objects.

2.2.2.2 Dynamic Objects

COMDALE/X has the capability to select from a class the appropriate object(s) that satisfies a particular condition during runtime. You can use this capability with dynamic objects and dynamic object set functions such as ANY and ALL.

Syntax:

`?objectname`

where `objectname` is the name of the dynamic object. The dynamic object is determined during inference and COMDALE/X substitutes the object(s) chosen whenever it sees a question mark (?) preceding the name of the dynamic object. The dynamic object could be assigned values using this syntax:

`?dynamic = ANY (condition | condition & ...)`

where `|` and `&` denote OR and AND respectively

Example:

`?x_set = ANY ({tank}.level.high is TRUE | {tank}.level.@f > 100.0)`

In this statement, COMDALE/X will examine the objects in the class tank to locate a tank which has a level high or level greater than 100. The ANY set function will return an object name(s). Therefore ?X_set dynamic object variable will be assigned with an object name(s) if the object(s) satisfied the conditions given in the ANY function.

NOTES:

1. In this example, tank is the class name, so that {tank} refers to the objects in the class.
2. In a set function such as ANY, you can have a list of object names, a subset of a class or a merger of two or more classes.

Example:

? tank = ANY ({tank}.level.@i > 1 | {bigtank}.level.@i > 20)

Dynamic objects can also be in expressions which evaluate to make assignments to other dynamic objects.

Example:

? specialtank = ANY (?tank.level.@f <= 40)

Example: A dynamic object can be a member of the class watertank with level greater than 25. This object is expressed in a rule as:

?dynamicictank.level.high

where ?dynamicictank = ANY ({ watertank }.level.@float > 25)

An object may be comprised of multiple smaller objects. For example a car may be made up of an engine, door, body, etc. These contributing objects are called **subobjects**.

2.2.3 Classes

COMDALE classes are used to group objects with similar attributes. Classes are created by the developer of knowledge bases to support some hierarchical or schematic representation of facts, or groups of facts, which are used to reason about a domain.

Example: Within the plant there exists many water tanks of which tank150 is one. Other water tanks may be tank10, tank30, etc. We can say that all of these objects, tank150, tank10 and tank30 belong to the same class; water tank.

By defining a class, we can list all the common attributes of these objects. Subsequently, if we define an object to be a member of a certain class, that object automatically will acquire or **inherit** the attributes of that class. By defining classes, we have the ability to reason about groups of objects without knowing the specific members of the group.

Note: An object may belong to classes. The combination of class, attribute and value is also called a keyword triplet, and has a set of facets associated with it.

Classes group objects with common attributes.

Example Each class may have multiple subclasses and may also belong to multiple classes. The hierarchy of classes is: Superclasses, classes and subclasses. Objects may belong to any of these classes.

Example: A “car”. This class may have attributes such as “price”, “horsepower”, “mileage” and “manufacturer”. Every member of this class would automatically be given these attributes. Members of the class “car” may be specific cars such as “Chevrolet_camaro”, or groups such as “Chevrolet_cars”. The specific cars are known as **objects** while the groups are called **subclasses**.

Inheritance Inheritance of a class attributes occurs when a new class is made to be a subclass of that class or an object is assigned to be a member of the class.

Note: If you remove a class from being a subclass, objects which are members of the removed subclass do not lose the attributes previously inherited from the class.

Public and private attributes Attributes of a class can be public and private. Public attributes are inherited by all subclasses and objects which are members of that class. Private attributes are only inherited by objects that are members of that class, subclasses do not inherit private attributes.

Creating classes Within COMDALE/X the class editor is used to define classes. Each class has a unique name and a set of attributes. The first step in defining classes is to use the editor to give the class a name then list its attributes. To create a subclass, you must first define it, then type its name in the subclass list of the desired class. The subclass will then inherit all the public attributes of the superclass.

Not all objects belong to a class Objects in a COMDALE/X knowledge base do not necessarily have to belong to a class. Therefore, it is not required that a hierarchy of classes and objects be developed or specified for each knowledge base.

“Class Editor” Window The Class Editor is used to create, maintain and delete COMDALE/X classes. It can be found under the **Editor** button in the main bar menu.

Class keyword-triplets A class keyword-triplet is the combination of class.attribute.datatype (or value). These keyword triplets are used to manipulate classes in COMDALE/X Expressions.

Examples:

```
{ car }.color.blue  
{ car }.horsepower.@float
```

Using classes in expressions Class names when used in COMDALE/X expressions are usually in braces. Functions used to manipulate classes are ANY () and ALL ().

Examples:

```
IF ANY ({car}.color.blue is TRUE)
```

OR ALL ({ car }.horsepower.@float > 120)

Where:

Example1: check if there are any objects in the class 'car' with attribute 'color' and value 'blue' that are TRUE.

Example: checks if all objects in the class 'car' with attribute 'horsepower' have floating point values which are greater than 120.

Note: Dynamic objects can be used to identify class members whose attribute.value meets specific criteria.

Classes and objects can belong to multiple classes.

2.2.4 More about Classes

In addition to objects being members of classes, classes themselves can be members of other classes. classes which are members of other classes are called **subclasses**. A **superclass** is one which has members that are classes.

Example: subclasses watertank and oiltank can be members of a superclass, storagetank, which could be a subclass of a superclass, tank.

Classes and objects automatically inherit the attributes of the class to which they belong. A class can have attributes which are inherited only by objects but not by subclasses of that class. These attributes are referred to as **private attributes** as opposed to **public attributes, which are inherited** by all subclasses and member objects.

Note: Classes and objects have no restrictions with respect to the number of classes in which they can have membership.

Rules are IF-THEN-ELSE representations of knowledge.

2.2.5 Rules

Rules represent reasoning knowledge and handle complex relationships between facts. Rules can embody vague concepts, simple heuristics, mathematical expressions, date expressions, time expressions, character string expressions or functions. Rules are written in an **IF-THEN-ELSE** format.

Each rule may have multiple conditions to be satisfied in the **IF** part (called the premise) and multiple actions or assignments in the **THEN** or **ELSE** part (called the conclusion).

Rule Syntax

Syntax:

if <expression> [& <expression>] [| <expression>]
and/or < expression> [& < expression>] [| < expression>]
and/or ...

then < expression> [cf = x]
then ...

else < expression> [cf = x]
else ...

if < expression>
and < expression>
or < expression>

are called condition statements.

then < expression>
else < expression>

are called conclusion statements.

cf = x is the certainty factor of the conclusion statement. This is assumed to be 100 if none is stated.

- There must be at least one condition statement in a rule, the first statement must begin with IF.
- Condition statements may be logical, string, numerical, date or time type expressions, and must examine a relation e.g. a triplet is true or a numerical expression is greater than or equal to another numerical expression, or execute a function. See chapters on “statements and expressions” and “functions” for valid syntax.
- There is no limit to the number of condition statements which you may use in a rule
- There must be at least one conclusion statement in a rule
- Conclusion statements may be logical, string, numerical, date or time type an assignment or execute a function. See chapter on “statements and expressions” and “functions” for valid syntax.
- There is no limit to the number of conclusion statements which you may use in a rule.

The certainty factor is a value which ranges from 0 to 100 and is a measure of how sure you are that the conclusion statement is true. If this is not specified it is assumed to be 100.

Example: In the plant, a rule can be written to close valve8 when the level of tank 150 is low and its temperature is high.

```
IF tank150.level.low  
AND tank150.temperature.high  
THEN text (“Please shut valve8”)
```

ELSE text (“Tank150 level is OK”)

The IF-AND-OR part of the rules is called the **premise**.
The THEN-ELSE part is called the **conclusion**.

In the premise, the statements after IF, AND, and OR are termed **condition statements**.

In the conclusion, the statements after THEN and ELSE are called **conclusion statements**. AND or OR are called **logical connectives** in COMDALE/X.

Condition and conclusion statements are used to state relationships between **expressions**.

COMDALE/X expressions consist of **keyword triplets, constants, functions** and **operators** and evaluate to logical, string, mathematical, date and time values. Each expression also returns a **degree of truth**. This degree of truth could be TRUE (100), FALSE (0) or some intermediate value, and reflects the confidence in the value of the expression. Constants evaluate to the value they express, e.g. TRUE, 10 and “high” are logical, mathematical and string constants respectively, cos (25) and text (“hi”) are functions and +, -, * are operators.

Example of a rule:

```
IF tank.level.@integer+5 > tank.height.@integer
OR tank.temperature.high
THEN text (“Please shut valve8”)
THEN tank.valve.@float = 25 + cos (40)
ELSE tank.valve.@float = 2
```

The premise of the rule is:

```
IF tank.level.@integer+5 > tank.height.@integer
OR tank.temperature.high
```

The conclusion of the rule is:

```
THEN text (“Please shut valve8”)
THEN tank.valve.@float = 25 + cos (40)
ELSE tank.valve.@float = 2
```

Condition statements are:

```
tank.level.@integer +5 > tank.high.@integer
tank.temperature.high
```

Conclusion statements are:

```
tank.value.@float = 25 + cos (40)
tank.value.@float = 2
```

Expressions are:

```
tank_level.@integer + 5
tank.height.@integer
```

$$\frac{25 + \cos(40)}{\cos(40)} \\ 2$$

Rules also have attributes which describe their characteristics. These include name, priority, the time interval between use, explanations, how uncertainty should be handled, the rule set to which this rule belong, etc. These attributes are configured during development of the knowledge base.

Example

Example of a rule:

```
IF tank.level.high is TRUE  
THEN tank.valve.shut is TRUE
```

This rule dictates that the keyword triplet “tank.valve.shut” will be set to be TRUE whenever the triplet “tank.level.high” is TRUE.

Rule Execution

During inference, the selection of a rule for execution depends on the conflict resolution mechanism in use at the time (see inference strategies).

At the start of consultation, rules may fire in alphabetical sequence according to their names. This can be customized using the start strategy and conflict resolution (if rule priorities are defined).

Rule execution begins with the first statement in the condition and progresses to the conclusion. If a rule is successful (i.e the net degree of truth is greater than the confidence level), the THEN part of the rule is executed, if not, the ELSE part will be executed (if it is defined).

In each expression of a rule, if COMDALE/X encounters a keyword triplet that is in NOTKNOWN state, and it is a subgoal or goal, it will try to backward chain to instantiate this triplet. If it is an input triplet COMDALE/X will ask the user a question, the presentation of the question can be customized using the question, exclusive, multichoice, etc. facet of the triplet. The DataSource facet of a keyword triplet can be used to override the default mechanism for instantiating triplets.

Rules are examined until the finish strategy criteria is satisfied.

Creating Rules

Within COMDALE/X the rule editor is used to define rules. Each rule has a unique name with optional attributes. The first step in creating rules is to use the editor to give the rule a name. Next move the cursor to the ‘Rule field’ and type the rule starting with IF.

Hint for creating Rules

In a very well developed knowledge base each rule should be a complete entity on its own, i.e. when you read a rule its meaning must be clear and exact.

Creating Rulesets

You may group rules which you have already created with the rule editor into rulesets. This is done using the Ruleset Editor within COMDALE/X.

“Rule Set Editor” Window

The “Rule Editor” is used to create, modify and delete, rules. It can be found under the **Editor** button on the main bar menu.

The “Rule Set Editor” is used to create, maintain and delete rulesets. It can be found under the **Editor** button in the bar menu.

1. IF TRUE
THEN DISPLAY (“hytext”, “topic”)
2. IF tank.level.high
THEN tank.level.open is FALSE
3. IF FORM (“name.frm”)
AND DISPLAY (“file”)
AND ANY ({ car }.horsepower.@f > 600)
THEN text (“this rule is successful”)
ELSE text (“this rule failed”)

Rules may belong to rulesets.

2.2.6 More about Rules

In addition to reasoning about individual objects, COMDALE/X rules can represent relationships between classes of objects or various facets of classes.

Example :

“If any member of the watertank class has a level that is low and that watertank temperature is high then close the valve to that tank”

```
IF ?tank = ANY ({ watertank }.level.low is TRUE &
{ watertank }.temperature.high is TRUE)
THEN ALL (?tank.valve.close is TRUE)
```

-OR-

Example:

“If we are more than 70% sure that tank150 level is low then examine all rules in the knowledge base that deal with a low level as well as open all tank valves”.

```
IF CERTAINTY (tank150.level.low) > 70.0
THEN APPLYRULE (“*.level.low)
THEN ALL ({ tank }.valves.open is TRUE)
```

Within a knowledge base, rules may belong to various **rule sets**. This is analogous to objects belonging to classes. A rule belonging to a set can inherit the attributes of that set. These rule attributes include, priority, name, explanation and methods of managing uncertainty. The main rule set is called the **root set** and is the default.

Procedures are like conventional programs.

2.2.7 Procedures

A procedure is a collection of instructions pertaining to rules, classes or objects, which perform special functions during the inference process.

Procedures can be used to sequence the execution of expressions or functions and other activities in a manner similar to conventional programming, in a predefined order.

Procedures contain one or more statements (boolean, date time, function etc.) which are executed one at a time starting from the top. Procedures can be used for batch processing as in the case of performing a recipe. Procedures can be evoked at any time during inference.

Example:

- 1 Find all tanks with low levels.
- 2 Execute all rules with tank.level.low
- 3 Pause for 30 seconds
- 4 Set tank150 temperature to 60C
- 5 Change the inference strategy to breadth search
- 6 Display text (“Tank150 temperature set”)
- 7 Run another procedure called “check temperatures”
- 8 Stop.

Example

Example:

```
ASK (“What is the value of the tank level?”,  
tank.level.@float)  
MACRO (“rule 5”)  
FIND (“tank.level.high”)  
APPLY RULE (“tank.valve.*”)
```

This procedure dictates a sequence of five functions which will be executed, in the order, top to bottom whenever this procedure is evoked.

In many cases procedures are used to initialize variables, present introductions and forms at the start and end of a consultation session.

Procedure Syntax

A procedure comprises of a list of any valid COMDALE/X expressions and functions.

Note: IF-THEN and IF-THEN ELSE constructs **cannot** be used in a procedure.

Procedure Execution

Procedures may be executed by naming them in the start strategy, report strategy or by executing the **run_procedure** function. Execution of a procedure starts with the first expression and proceeds to the last.

During the execution of a procedure, if a triplet which is NOTKNOWN is encountered, COMDALE/X will backward chain or ask a question, in manner

similar to the execution of rules. If you simply list various input keyword triplets in your procedure, COMDALE/X will automatically ask questions to instantiate them if they are not known.

Creating Procedures

Within COMDALE/X the procedure editor is used to define procedures. Each procedure has a unique name. The first step in creating a procedure is to use the editor to give the procedure a name then type your list of functions and expressions in the “DO field”.

“Procedure Editor” Window

The “Procedure Editor” is used to create, modify and delete procedures. It can be found under the **Editor** button on the main bar menu.

2.3 Inference Engine

Control strategies:

The COMDALE/X Inference Engine uses a **compiled** version of the knowledge base during inference.

Note: The knowledge base is not precompiled prior to use by COMDALE/X, it is compiled while being loaded.

The Inference Engine processes the knowledge contained in the knowledge base to make decisions. It contains the methodology for reasoning, which in COMDALE/X is comprised of four control strategies:

Start

Start is used to dictate what is to be done when starting the inference process. With this strategy a procedure can be executed, variables initialized, the inference engine configured, etc.

Inference

The inference strategy guides the activities of the inference engine during the inference process. This can be used to customize how the inference engine handles forward and backward chaining, as well as depth and breadth first approaches to reasoning, and knowledge base search. In addition, conflict resolution mechanisms can be customized.

Finish

The finish strategy is used to dictate when the inference process should terminate.

Example: When all rules have been examined.

Report

A report strategy dictates what should be done after the inference process has terminated.

Example: Execute a procedure, show all conclusions, restart, print reports, etc.

In managing a consultation session, the inference engine will look at the finish strategy to find out how and when the session is to end. Armed with the finish strategy criteria, the inference engine then applies its start strategy to being the inference process. The inference strategy is then used by the inference engine to guide both the **forward** and **backward chaining** activities of the inference process.

Forward chaining is a search procedure or reasoning process using known facts to produce new facts and reach a final conclusion. On the other hand, backward chaining is a reasoning process which starts with a desired goal and works backward, looking for facts and rules that support the desired outcome.

The inference engine, conflict resolution and focus mechanisms are called upon in the inference strategy to resolve any conflicts, such as which rule is to be used first when there is more than one rule eligible for execution at that time.

Upon termination of the inference process, the inference engine will apply its report strategy. At any time during the consultation process these strategies can be modified.

The control strategy is kept in a file with the same name as that of the knowledge base and has extension **.stg**.

**User Interface
comprised of bar
menus, graphical
editors, icons etc.**

2.4 User Interface

The user interface is the facility which accommodates communication between the user and COMDALE/X. The user interface can be **graphical** or **character based**.

For development of the knowledge base, **graphical editors** such as the class, object, rule, and procedure editors can be used. You can also use any ASCII editor of your choice. Facet editors are available as pop ups from the class and object editors.

To assist in the development effort, many views of the knowledge base are available. These include a **class/object browser** which illustrates the hierarchical relationship between classes and objects, and a **rule browser** which shows the interrelationship between rules. COMDALE/X allows you to edit your knowledge base from these browsers.

Conceptually, the development interface is **consistent** yet provides a multidimensional view of the knowledge base. All editors and browsers are accessible from **pull down menus** and **icons**.

During debugging, the user interface accommodates cross referencing of keyword triplets, rule tracing, watching keyword triplets as they are instantiated, as well as session recording and replay facilities.

The COMDALE/X inference engine can be **embedded** in an application thus having no interface, or can be customized by the developer.

For consultation, a user can utilize COMDALE/X in many ways, these include:

- 1 Volunteering information then asking the system to delivery its findings.
- 2 Asking the system to verify or test if various hypotheses are true or false.
- 3 Asking the system to execute, ask questions, then deliver findings.

At any point during the consultation one can ask for an explanation, justification, or for general information about the knowledge contained in the knowledge base.

Using the graphical user interface you may be use COMDALE/X to:

1. Solve problems
2. Provide information
3. Explain and justify actions

Utilities include hypertext encryption, and forms.

2.5 Utilities

COMDALE/X packages a number of utilities to facilitate various approaches to representing and displaying information, as well as to provide security for the knowledge base.

The **Hypertext** utility is included to provide on-line interactive documentation during a consultation session.

The **Encryptor** utility is used to encrypt the knowledge base for security.

The **Forms** utility is used to generate forms for input to COMDALE/X

2.6 Keyword triplets

The object, attribute and value parts of a keyword triplet are called tokens.

The keyword triplet in COMDALE/X represents the main building block in the assembly of the knowledge base. It is used by COMDALE/X to represent factual information. Keyword triplets are made of either an **object.attribute.value** or **class.attribute.value** combination.

An object is a physical or conceptual entity. A class represents relationships between similar objects or classes in a hierarchical and schematic way. An attribute is a property associated with an object or a class. An object or class may have multiple attributes, each attribute takes different values.

There are five types of keyword triplets, depending on the value type of the attribute, they are:

- logical (or boolean)
- numerical
- string
- date
- time

An object attribute or class attribute can have more than one value type.

Keyword triplets other than those of logical (Boolean) type can have their values changed. These are referred to as **triplet variables**.

Logical keyword triplet e.g. tank.level.high

Logical keyword triplets have an attribute with a logical value.

Examples of logical values:

high, low, ok, trending_up

Examples of logical keyword triplets:

tank.level.high
line.temperature.high
copper.assays.trending_up

**Numerical
keyword triplet
e.g.
tank.level.@i**

A numerical keyword triplet has an attribute which has a numerical value. There are four types of numerical keyword triplets:

float the value token can be specified as **@float** (or **@f**), which denotes that the value of the keyword triplet is a single precision real (floating point) number.
Double **@double** (or **@db**) can be used to specify the value token, which denotes that the value of the keyword triplet is a double precision floating point number.
integer the value token can be specified as **@integer** (or **@i**) to denote that the value of the keyword triplet is a two byte (short) integer.
long the value token can be specified as **@long** (or **@l**) to denote that the value of the keyword triplet is a four byte (long) integer.

Examples of numerical keyword triplets:

tank.level.@f
tank.temperature.@double
pressure.reading.@integer

**String keyword
triplets e.g.
tank.level.@s**

A string keyword triplet has an attribute which has a string value. These keyword triplets are represented in COMDALE/X with the value token, @s or @string.

Examples of string keyword triplets:

tank.level.@s
interest.rate.@string

**Date keyword
triplets e.g.
ship.date.@dt**

A date keyword triplet has an attribute which has a date value. These keyword triplets are represented in COMDALE/X with the value token, @date or @dt. Dates are stored as year, month, day.

Examples of date keyword triplets:

tank.manufacture.@date
tom.birthday.@date

**Time keyword
triplet e.g.
wait.time.@t**

A time keyword triplet has an attribute which has a time value. These keyword triplets are represented in COMDALE/X with the value token, @time or @t. Times are stored as hour, minute, second.

Examples of time keyword triplets:

race.start.@time
system.reboot.@time

2.6.1 Input Keyword Triplets

If a triplet is used in the premise of a rule and not in the conclusion of another rule it is called an **input triplet**.

In order for the value to be known to the inference engine, the expert system queries the point database. When using COMDALE/X, the expert system will query the user for this information.

2.6.2 Subgoal Keyword Triplet

Subgoal Keyword Triplets are assigned values and certainties in the conclusion of rules **and** are also used in the premise of other rules. The inference engine will backward chain to instantiate these triplets.

Note: The value and certainty of subgoal triplets are modified in the point database. Each time they are assigned a new value by the inference engine.

If the rule that includes the subgoal triplet fails because the premise statements fail to become instantiated, the subgoal triplet will be assigned UNKNOWN, even though the triplet may have a higher degree of certainty within the point database. The developer should use the ASKCC or GETPOINTDATA functions to force the inference engine to look for the triplet value and certainty in the point database.

2.6.3 Goal Keyword Triplets

Goal Keyword Triplets are assigned a value and certainty in the conclusion of rules but are not used in the premise of other rules. Values and certainties of goal triplets are modified in the point database each time they are changed by the inference engine.

2.7 Triplet Facets

Facets are used to describe some property of a keyword triplet, and are used to specify a wide variety of actions or functions related to the keyword triplet. These ranges from customizing the question that is asked when COMDALE/X needs to instantiate the triplet, to defining a fuzzy set which quantifies a qualitative triplet (e.g. tank.color.red).

Example:

The following function may be specified in the 'question' facet of the keyword triplet "tank.level.high".

ASK ("Is the tank level currently high", tank.level. high)

This function will then be executed whenever COMDALE/X attempts to instantiate this triplet by asking a question.

Editing Facets

Facets of a keyword triplet are created, modified using the **Facet Editor**. This editor is accessed from the class and object editors by selecting the check box beside the desired attribute field of the object or class.

Question Facet allows you to customize a question to the user

2.7.1 Question Facet

The ‘**Question facet**’ allows you to customize a question to the user. The answer supplied by the user will be used to instantiate the corresponding keyword triplet.

The Question facet is used in COMDALE/X, and is not used in COMDALE/C.

The function **ask ()** is used to customize a question.

Example:

ASK (“What is current tank level?”, tank.level.@float)

This question will be asked when the value of the triplet tank.level.@float is required.

Note: More than one function can be used in the question editor.

If a Question facet is not configured in COMDALE/X cannot instantiate the triplet from any rules in the knowledge base, a question will be automatically generated by COMDALE/X, utilizing the triplet name as the basis of the question.

Examples:

1. **tank.level.@float**
“What is the value of tank level?”
2. **tank.level.high**
“Is tank level high?”

2.7.2 Explanation Facet

The ‘**Explanation facet**’ allows you to customize an explanation of a question that is accessible to the user from an **Explanation** button in the question window.

The Explanation facet is used in COMDALE/X, and is not used in COMDALE/C.

Typically the functions **text ()**, **paint ()** and **display ()** are used to provide an explanation.

Example:

TEXT (“This means that the tank level is above 20ft”)

Note: More than one function can be used in the explanation editor.

Explanation Facet allows you to customize an explanation of a question that is accessible to the user from an Explanation button in the question window

Default Facet allows you to assign a default value to a keyword-triplet

2.7.3 Default Facet

The '**Default facet**' allows you to assign a default value to a keyword-triplet.

The Default facet is used in COMDALE/X and COMDALE/C.

Whenever COMDALE/X needs to find information about a keyword-triplet which has not been previously instantiated, it first checks if this keyword-triplet can be instantiated from within the knowledge base.

If this keyword-triplet is available for instantiation in the conclusion of a rule, COMDALE/X then tries to fire this rule. If this and all other rules involving the keyword-triplet fail, COMDALE/X then assigns the default value to this keyword-triplet or in the case of a boolean the default degree of certainty.

Note: If the triplet is an input and the user answers unknown to a question the default will also be applied.

Accumulation Facet is used to determine how the degree of certainty of a triplet is accumulated

2.7.4 Accumulation Facet

The '**Accumulation facet**' is used to determine how the degree of certainty of a triplet is accumulated, i.e., if multiple rules in a knowledge base attempt to instantiate a triplet, accumulation dictates the final degree of certainty that would be assigned to the triplet.

The Accumulation facet is used in COMDALE/X and COMDALE/C.

The following options are available:

\$MAX - the highest degree of certainty.

\$MIN - the lowest degree of certainty.

\$AVG - the average degree of certainty.

\$ANY - the first degree of certainty that is greater than the confidence level of the system.

Restriction facet is used to specify legal values that a keyword triplet may be assigned

2.7.5 Restriction Facet

The '**Restriction facet**' is used to specify legal values that a keyword triplet may be assigned.

The Restriction facet is used in COMDALE/X and COMDALE/C.

The constraint may be expressions in conjunction and disjunction. To put expression in conjunction use & for disjunction use |.

Examples:

1. tank.level.@float > 10 & tank.level.@float < 100

means that tank.level.@float must be between 10 and 100

2. tank.level.@float > 10 | tank.level.@float < 1

means that tank.level.@float must be either greater than 10 or less than 1.

DataSource facet is used to override the default order in which COMDALE/X goes about instantiating keyword triplets

2.7.6 DataSource Facet

The **'DataSource facet'** is used to override the default order in which COMDALE/X goes about instantiating keyword triplets. You may specify functions such as find (), ask () or others to get values from databases and spreadsheets.

The DataSource facet is used in COMDALE/X and COMDALE/C.

Whenever COMDALE/X needs to find information about a keyword-triplet which has not been previously instantiated, it first checks if this keyword-triplet can be instantiated from within the knowledge base.

If this keyword-triplet is available for instantiation in the conclusion of a rule, COMDALE/X then tries to fire this rule.

If this keyword-triplet is not available for instantiation in the conclusion of a rule, or the value cannot be determined from the firing of any rules:

- In COMDALE/X a question will be asked of the user;
- In COMDALE/C the inference engine will query the database for the value of the triplet.

Using the **'DataSource facet'** you can override the default order in which COMDALE/C or COMDALE/X goes about instantiating keyword triplets.

Example:

ASK ("What is the tank level?" tank.level.@float)

can be used to ask a question to instantiate a subgoal.

2.7.7 IfChange Facet

The **'IfChange facet'** is used to specify various functions which will be executed when the value of a keyword triplet is changed.

The IfChange facet is used in COMDALE/X and COMDALE/C.

2.7.8 Fuzzy Set Facets

A **'Fuzzy set facet'** is used to give a logical variable a degree of certainty which depends on the numerical value of an expression. This expression is called the source and its value is mapped to defined degree of certainties. With this feature, vague concepts such as high, low, and "a little while ago", can be represented.

The Fuzzy Set facet is used in COMDALE/X and COMDALE/C.

Note: Fuzzy set when used with the change () function may have the numeric 1 as a source.

IfChange facet is used to specify various functions which will be executed when the value of a keyword triplet is changed

Fuzzy Set facets are used to give a logical variable a degree of certainty which depends on the numerical value of an expression

Exclusive facets will enable a list of option values of which only one can be selected

2.7.9 Exclusive Facets

An **'Exclusive facet'** will enable a list of mutually exclusive values for the attribute of an object of which only one value can be selected.

The Exclusive facet is used in COMDALE/X, and is not used in COMDALE/C.

Only one value can be instantiated as TRUE (with whatever net degree of truth is specified by the rule or procedure), all other values will be set to FALSE (with a degree of truth = 100 – degree of truth of TRUE value).

Note: If you define both an exclusive and a multichoice facet for the same group of object attributes then the exclusive set will take preference and the multichoice list will be ignored.

2.7.10 MultiChoice Facets

MultiChoice facets will enable a list of mutually exclusive values for the attribute of an object of which many values can be selected

A **'MultiChoice facet'** will enable a list of mutually exclusive values for the attribute of an object of which many values can be selected.

The MultiChoice facet is used in COMDALE/X, and is not used in COMDALE/C.

Any selected values will be instantiated as TRUE (with whatever net degree of truth is specified by the rule or procedure), all other values will be set to FALSE (with a degree of truth = 100 – degree of truth of TRUE value).

Note: If you define both an exclusive and a multichoice facet for the same group of object attributes then the exclusive set will take preference and the multichoice list will be ignored.

Facet Inheritance

All facets can be defined at the class keyword triplet level. All member objects can then inherit those facets by enable this facility in the facet editor.

2.8 Managing uncertainty

In the 'real' world, situations are not always clear cut, there are usually uncertainties. COMDALE/X recognizes this and has appropriate features to deal with uncertainty. COMDALE/X also allows the application developer to define how it should deal with uncertainty. The following discussion focuses on uncertainty in the COMDALE/X environment.

2.8.1 Uncertainty

Each triplet has a degree of certainty.

Each keyword triplet has a facet called **degree of certainty (dc)**. The degree of certainty, a numerical value which ranges from 0 to 100, is a measure of how sure the system is that the value of the attribute is true.

True and False

TRUE and **FALSE** are logical constants which evaluate to degrees of certainty of 100 and 0 respectively. A degree of certainty of 0 means that the value is FALSE while a value of 100 means that the value is TRUE.

Example:

tank.level.high dc = 70%

This means that we are 70% sure that the tank level is high.

tank.level.@integer = 20 dc = 90%

We are 90% sure that the tank level is equal to 20.

During inference the degree of certainty of a keyword triplet can be **NOTKNOWN**, **KNOWN** or **UNKNOWN**.

- **NOTKNOWN** is before inference commences.
- **KNOWN** is when it is assigned a degree of certainty (which ranges from 0 to 100, 0 meaning FALSE, 100 meaning TRUE).
- **UNKNOWN** is when COMDALE/X tried unsuccessfully to make the triplet **KNOWN**.

When the degree of certainty of a keyword triplet is known it is said to be **instantiated**.

To decide if rules are successful (i.e the “then” part will be examined), or unsuccessful (the “else” part examined), COMDALE/X evaluates a measure of how sure it is that the premise of the rule is true. This measure is called the **net degree of truth**. If the net degree of truth is greater than, or equal to the globally defined **confidence level**, the rule is considered to be successful.

In rules, conclusion statements (the “then” and “else” statements) may have a **certainty factor** (cf) attached to each statement. This certainty factor ranges from 0 to 100 and indicates the confidence that you have in that conclusion statement. By default, cf equals 100. This reflects uncertainty in knowledge. When executing a rule during inference, COMDALE/X combines degrees of certainty and certainty factors to derive new degrees of certainty of keyword triplets in the conclusions of rules.

Example:

IF tank.level.high

THEN valve.position.@float = 25 cf = 90

The cf is the certainty factor for this conclusion statement. In this case it demonstrates that we are 90% sure that the valve position must be set to 25 if we are 100% sure that the tank level is high.

Note: During inference the degree of certainty of the valve position will depend on the degree of certainty of tank.level.high as well as the certainty factor.

Instantiation means obtaining a degree of certainty.

2.8.2 Methods of Instantiation

Keyword triplets can be instantiated in many ways. For example, by asking the user a question, by the user volunteering information, by assignment in the conclusion of a rule, by default, by accessing a database, spreadsheet, etc.

During a consultation session, in order for expressions to be evaluated, the keyword triplets in the expressions must be instantiated. COMDALE/X does this on an as-is-required-basis. This is, as the expression is being evaluated and the keyword triplet is encountered, COMDALE/X will attempt to instantiate the triplet, if its degree of certainty is not already known.

The default method COMDALE/X uses to instantiate keyword triplets is as follows:

If the keyword triplet is a goal or subgoal, COMDALE will backward chain to examine rules which conclude about this triplet. The selection of rules will be decided by the inference strategy being applied. If after backward chaining the keyword triplet is still not instantiated, it is then assumed to be UNKNOWN unless the default facet of the triplet has been specified. In such a case the default will be used.

If the keyword triplet is an input variable COMDALE/X will ask the user a question. The nature of the question will dependent on the keyword triplet type as well as if it has a fuzzy set, or is part of a mutually exclusive set, etc. If the user answers UNKNOWN to a question and the default facet has been specified, the triplet will be assigned the default degree of certainty and value.

The datasource facet of a keyword triplet can be used to specify the order in which specific attempts are made to instantiate a keyword triplet.

2.8.3 Accumulation of Certainty

During consultation, a keyword triplet may already be instantiated, yet through assignment, database access etc. the system may attempt to assign a new degree of certainty to the triplet. By default, COMDALE/X will maintain the lowest degree of certainty. You can customize the method of accumulation of certainty by using the accumulation facet of a keyword triplet. For example, you may assign the average degree of certainty or choose the maximum.

An example of how accumulation could be used is when a subgoal keyword triplet appears in the conclusion of many rules and all of these rules are evaluated, each attempting to assign a different degree of certainty to the triplet. The accumulation facet could be used to manage the assignment of the final degree of certainty to the triplet.

Accumulation of degrees of certainty

Each conclusion statement has a certainty factor.

2.8.4 Uncertainty In Rules

Each conclusion statement in a COMDALE/X rule is accompanied by a certainty factor.

The certainty factor ranges from 0 to 100 and indicates the confidence you have in that conclusion statement, that is, it is a measure of the uncertainty in knowledge. The value of the certainty factor reflects the confidence in that conclusion statement, if the premise of the rule were 100% true.

Example:

IF tank.level.high
THEN tank.valve.shut is TRUE cf = 90

The rule indicates that if we are 100% sure that the tank level is high we are only 90% sure that the tank valve is shut.

2.8.5 Evaluating Rules

Evaluating Rules

In evaluating rules, the following concepts are used to determine if a rule will be successful or will fail:

- Degree of truth
- Net degree of truth
- Confidence level

During the inference process, COMDALE/X evaluates each statement in the premise of a rule to determine a degree of truth of each statement. The degree of truth is how sure COMDALE/X is that the statement is true, (based on the values and degrees of certainty of the keyword triplets and other elements of the statement). The degree of truth ranges from 0 to 100, where 100 indicates TRUE and 0 FALSE. Intermediate values are measures of various degrees of truth. The method of calculating degrees of truth varies with the type of statement. The degrees of truth of each statement of a rule is combined to determine a **net degree of truth** of the rule. This is a value which ranges from 0 to 100 and specifies how sure COMDALE/X is that the premise of a rule is true.

For each rule the **net degree of truth** is compared with the **confidence level** to determine if a rule will be successful or will fail. **A rule only be successful if the net degree of truth is greater than or equal to the confidence level.** The **then** conclusion statements of a rule are examined only when the rule is successful. The **else** conclusion statements of a rule are examined only when the rule fails.

The **inference attribute** of a rule can be used to customize how the degrees of truth are calculated and combined in a rule.

2.8.6 Logical Connectives

Logical Connectives and net degree of truth.

The **logical connectives (and, or)** determine the relationship between conditions which must exist before the conclusion of a rule is evaluated. COMDALE/X uses logical connectives to evaluate the net degree of truth. The net degree of truth of a rule is the result of the relationship between the degree of truth of each condition statement in the rule and the logical connectives which link these conditions.

**AND – minimum
(by default)**

When conditions are ‘anded’, by default, the minimum of the degrees of truth of conditions is taken as the net degree of truth of these conditions.

When conditions are **ored**, by default, the maximum of the degrees of truth of conditions is taken as the net degree of truth of these conditions. COMDALE/X, allows customization of the method for the evaluation of both ‘and’ and ‘or’ in rules

is, equal

2.8.7 Assignment

In conclusion statements, the certainty factor is used to determine the degree of certainty of the keyword triplet on the left hand side of an assignment statement. The keyword triplet is assigned a degree of certainty which is equal to the product of:

- 1 The certainty factor
- 2 The net degree of truth
- 3 The degree of truth of the expression on the right hand side of the statement.

This product is divided by 10,000 (100 x 100) to give a degree of certainty which ranges from 0 to 100.

Note: The assignment is done depending on:

The accumulation facet of the keyword triplet

- 1 The rule is successful (the THEN part is assigned)
- 2 The rule fails (the ELSE part is assigned)
- 3 The inference attribute of the rule

**Uncertainty in
logical and string
expressions.**

2.8.8 Evaluating Degrees of Truth of Condition Statements

IS and IS_NOT are used in logical, string, date and time type rule condition statements to describe the relationship between values. **IS and IS_NOT are not case sensitive.**

Examples of logical and string statements:

- 1 tank.level.high is TRUE
- 2 if tank.level.high is not TRUE...
- 3 tank.level.@string is “high”
- 4 tank.level.@string is tank.height.@string

By default, COMDALE/X evaluates the degree of truth of an expression to be the minimum degrees of truth returned by all constants, functions and keyword triplets in the expression.

Note: The degree of truth of a keyword triplet is equal to its degree of certainty. The degree of truth of a function depends on the function used.

IS Statement

2.8.9 Logical

When “IS TRUE” is used in logical expressions the degree of truth is equal to the degree of certainty of the keyword triplet. When “IS FALSE” is used, the

degree of truth is equal to 100 minus the degree of certainty of the keyword triplet.

**IS NOT
Statement**

When “IS_NOT TRUE” is used in logical expressions, the degree of truth is equal to 100 minus the degree of certainty of the keyword triplet. When “IS_NOT FALSE” is used, the degree of truth is equal to the degree of certainty of the keyword triplet.

IS Statement

2.8.10 String

When “IS” is used in string expressions, the statement degree of truth is 0 if the expressions on both sides of the statement do not evaluate to the same string value. If they evaluate to the same string value, the degree of truth is equal to the minimum degree of truth of all expressions in the statement.

**IS NOT
Statements**

When “IS_NOT” is used in string expressions, the statement degree of truth is 0 if the expressions on both sides of the statement evaluates to the same string value. If they do not devaluate to the same string value, the degree of truth is equal to the minimum degree of truth of all expressions in the statement.

Examples:

Examples of condition statements using “IS” and “IS_NOT”:

tank.level.high is TRUE or tank.level.high

The degree of truth of the above statement is equal to the degree of truth (degree of certainty) of the logical expression (keyword triplet) tank.level.high

tank.level.high is FALSE or tank.level.high is_not TRUE

The degree of truth of the statement is equal to 100 minus the degree of truth (degree of certainty) of the logical expression (keyword triplet) tank.level.high

tank.level.@string is “high”

The degree of truth of this statement is 0 if the value of the string expression (string triplet variable) tank.level.@string is not equal to the string constant “high”. If the value is equal to “high”, the degree of truth is equal to the degree of truth (degree of certainty) of the expression (triplet) tank.level.@string.

tank.level.@string is_not tank.height.@string

The values of the triplets are compared, if they are the same, the degree of truth is equal to 0. If the values are not the same, the degree of truth is equal to the minimum degree of certainty of the triplets tank.level.@string and tank.height@string.

**Uncertainty in
mathematical
expressions.**

2.8.11 Relational Operators, Mathematical Expressions

A condition statement which tests a mathematical expression is constructed as follows:

expression1 relational operator expression2

Example:

tank.level.@float != 100 + tank.height.@float

By default, COMDALE/X evaluates the degree of truth of an expression to be the minimum degrees of truth returned by all constants, functions and keyword triplets in the expression.

Note: The degree of truth of a keyword triplet is equal to its degree of certainty. The degree of functions depend on the function used.

The following are the available relational operators for the conditions of rules.

<u>Relational Operators</u>	<u>Description</u>
==	equal to
!=	not equal to
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to

If the relation shown in the condition is false, then the condition statement is evaluated to a degree of truth of 0. If the relation is true, the condition statement is evaluated to a degree of truth equal to the minimum degree of truth of the expressions on the right and left hand sides of the expressions.

Note: The degree of truth of constants is assumed to be 100.

Examples:

tank.level.@float >= 100

Examples

The degree of truth of this statement is equal to 0 if the value of tank.level.@float is not greater than, or equal to 100. If the value is greater than, or equal to 100 the degree of truth of this statement is equal to the degree of certainty of tank.level.@float.

tank.level.@float >= 100 + tank.height.@float

The degree of truth is equal to the minimum of the degrees of certainty of tank.level.@float and tank.height.@float if the relation evaluates to be true.

2.9 Statements and Expressions

COMDALE/X expressions consist of **keyword triplets, functions and operators** and evaluate to logical, string, mathematical, date and time values. Each expression also returns a **degree of truth**. This degree of truth could be TRUE (100), FALSE (0) or some intermediate value, and reflects the confidence in the value of the value of the expression. Constants evaluate to the value they express. e.g. TRUE, 10 and “high” are logical, mathematical and string constants respectively. cos (25) and text (“hi”) are functions. +, -, * are operators.

COMDALE/X statements formulate relationships between expressions.

COMDALE/X permits five types of expressions:

- Logical
- String
- Mathematical
- Date
- Time

2.9.1 Logical Expression

Logical expressions evaluate to be TRUE or FALSE or some intermediate degree of truth. Statements can be used to test relationships between logical expressions or to assign a degree of certainty to a logical triplet.

Logical expressions evaluate to TRUE, FALSE or intermediate degrees of truth.

Examples of logical expressions:

TRUE
FALSE
tank.level.high
KNOWN (tank.level.high)

TRUE and FALSE are called logical constants.

Syntax:

logical expression **is** logical expression
logical expression **is_not** logical expression

Examples:

tank.level.high is TRUE
tank.level.low is_not TRUE

Note: In the premise of rules you may replace “tank.level.high is TRUE” with “tank.level.high”

When a logical statement is used in the condition of a rule it is a test. In the conclusion of a rule it is considered to be an assignment.

In the conclusion of rules the “is TRUE” or “is FALSE” must be stated to make an assignment.

String expressions evaluate to string values

2.9.2 String Expressions

String expressions evaluate to string values and may be string constants, string triplet variables, and functions which return string values. These expressions return string values and degrees of truth. The degree of truth could be TRUE (100), FALSE (0) or some intermediate value, and reflects the confidence in the value of the expression. These expressions may be used in statements for comparison of qualitative concepts, or used in assigning new string values to keyword triplets.

Examples of string expressions:

```
"low"  
tank.registry.@string  
attr (tank.level.high)
```

"low" is a string constant.

String constants can contain any number of characters. They must be bounded by double quotes.

Syntax:

```
string expression1 is string expression2  
or string expression1 is_not string expression2
```

Examples:

```
tank.level.@string is "high"  
tank.status.@string is_not "ok"  
user.name.@s is new.name.@s
```

When a string statement is used in the condition of a rule it is a test. In the conclusion of a rule it is considered to be in an assignment. In the conclusion of rules only the 'is' operator should be used.

Mathematical expressions evaluate to numerical values

2.9.3 Mathematical Expressions

Mathematical expressions evaluate to numerical values and may consist of constants, mathematical triplet variables, additive operators, multiplicative operators and function calls. These expressions return numerical values and degrees of truth. The degree of Truth could be TRUE (100), FALSE (0) or some intermediate value, and reflects the confidence in the value of the expression. Mathematical expressions allow COMDALE/X to perform mathematical computations in the condition and conclusion statement of rules as well as in procedure statements.

Syntax:

```
mathematical relational operator mathematical  
expression1 expression2
```

Examples:

```
tank.level.@float > 60  
60 + 2 >= tank.height.integer
```

When a mathematical statement is used in the condition of a rule it is a test. In the conclusion of a rule it is considered to be an assignment.

2.9.3.1 Additive Operators

Additive operators are evaluated from left to right.

Additive operators that are used in COMDALE/X are:

<u>Additive operators</u>	<u>Description</u>
+	Evaluates the sum of two expressions
-	Evaluates the difference of two expressions.

Syntax:

expression1 **additive operator** expression2

2.9.3.2 Multiplicative Operators

Multiplicative operators have a higher priority in their evaluation than the additive operators. A multiplicative operator is evaluated from left to right.

The following are the available multiplicative operators in COMDALE/X:

<u>Multiplicative operators</u>	<u>Description</u>
*	Evaluates the product of two expressions
/	Evaluates the division of expression1 by expression2
□	Evaluates expression1 raised to the power of expression2

Syntax:

expression1 **multiplicative operator** expression2

In a combination of operators, COMDALE/X uses the following order of priority in evaluating the mathematical expression:

<u>Operators</u>
()
□
*, /
+, -

You can use parentheses to group mathematical expressions according to the priority of operation.

The condition statement of any rule may be a test of the relation between two mathematical expressions. The following are the available relational operators in COMDALE/X.

<u>Relational Operators</u>	<u>Description</u>
= =	Equal to
!=	Not equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to

Syntax of a statements using mathematical expressions:
 expression1 **relational operator** expression2.

Note: In a relational test, all the operators in the expressions will be evaluated before any relational operator is used.

In COMDALE/X, there is only one mathematical assignment operator, = , and it is used to assign the value of the expression on the right hand side of a statement to the variable on the left hand side. The assignment operator '=' can only be used in the conclusion of a rule.

Examples of rules containing mathematical expressions:

```
IF area.of_circle.@f = = area.of_square.@f
OR specific_heat.of_fusion.@f <= 24.0
THEN characteristic.of_object.@s is "ok" cf = 90
```

```
IF percent_80.feed_size.@f <= 4000 * SQRT (13/word.index.@f)
AND grinding.power.@s is "for_ballmill"
OR percent_80.feed_size <= 15000 * SQRT (13/work.index.@f)
AND grinding.power.@s is "for_rod mill"
THEN efficiency.factor_4.@f = 1.0 cf = 100
```

```
IF boiler.temperature.@s is "high"
AND boiler.temperature.@f > 36
THEN boiler.pressure.@s is "to_be_checked" cf = 100
THEN boiler.temperature.@s is "very_high" cf = 100
```

2.9.4 Date Expression

Date expressions evaluate to date values.

Date expressions evaluate to date values and may be date constants, date triplet variables, and functions which return date values. These expressions return date values and degrees of truth. The degree of truth could be TRUE (100), FALSE (0) or some intermediate value, and reflects the confidence in the value of the

expression. These expressions may be used in statements for comparison of date concepts, or used in assigning new date values to keyword triplets.

Examples of date expressions:

this.date. @date
\$current_date

Where \$current_date is the value of the current date

Syntax:

date expression1 **is** date expression2
or date expression1 **is_not** date expression2

Example:

Printer.delivery.@**date** is \$current_date

When a date statement is used in the condition of a rule it is a test.
In the conclusion of a rule it is considered to be an assignment.
In the conclusion of rules only the 'is' operator should be used.

2.9.5 Time Expression

Time expressions evaluate to time values.

Time expressions evaluate to time values and may be time constants, time triplet variables, and functions which return time values. These expressions return time values and degrees of truth. The degree of truth could be TRUE (100), FALSE (0) or some intermediate value, and reflects the confidence in the value the expression. These expressions may be used in statements for comparison of time concepts, or used in assigning new time values to keyword triplets.

Examples of time expressions:

this.time.@time
\$current_time

Syntax:

time expression1 **is** time expression2
time expression1 **is_not** time expression2

Example:

Printer.delivery.@time is \$current_time

When a time statement is used in the condition of a rule it is a test.
In the conclusion of a rule it is considered to be an assignment.
In the conclusion of rules only the 'is' operator should be used.