Knowledge Expert

Preface

Knowledge Expert is a new generation product which allows users to build up and share corporate knowledge stored in rule bases, and leverage it across the company to ensure design compliance with established standards.

Adding to the native capacity of the Version 5 products and architecture to dynamically capture design specifications, Knowledge Expert delivers a way to:

- **Create and manage Generic Rules and Checks (P2 only)**
  Knowledge Expert enables users to define generic rules and checks specifications for classes of objects and store them in a base. These rules and checks can then be used to monitor the actions of every designer in the company. As geometry is created or changed, the system uses the rules and checks to ensure compliance to corporate standards.
  When a rule or check is violated, corrective actions can be recommended or automated using VBScript macros, texts or linked to URL files.

- **Manage and reuse corporate knowledge**
  Users can define and manage rule sets to structure the corporate knowledge base: Rules are then classified in rule sets that belong to a rule base. This structure allows different sets of rules and checks to be set up for different design or manufacturing processes according to the user's needs.
- With KWE, corporate knowledge can be shared throughout the company in rule bases that can be applied to models. Those rule bases are stored in documents that can then be imported.

- **Report with complete descriptions of the checks performed**
  KWE offers report capabilities in output formats such as HTML, XML or TXT permitting, for example, the publishing of customized reports of rules and check violations. Reports can be generated with short or long problem descriptions (depending on the level of detail required), and can include a list of all results or only the failed rules (or only the passed checks).

Conventions
Using this Guide

Using this Guide

This User's Guide is intended to help expert users become quickly familiar with the Version 5 of Knowledge Expert.

To get the most out of this Guide, it is highly recommended to start reading and performing the tasks described in the step-by-step tutorial, known as the Getting Started section and reading the Workbench Description to find his way around the Knowledge Expert Workbench.

This User's Guide is organized into the following sections:

- Preface: A short introduction to the product.
- What's new: A presentation of the new and enhanced product functions.
- Getting Started: A step-by-step tutorial intended to provide the user with an overview of the product functions.
- Basic Tasks: A description of the basis tasks as well of the Knowledge Expert language and tools.
- Advanced Tasks: A description of more advanced tasks.
- Workbench Description: A presentation of the user interface.
- Glossary: A list of terms specific to Knowledge Expert.

What's New?

No enhancements in this release.

Getting Started

See the Quick Reference section for a summary of the interactive tasks you can perform using the Knowledge Expert workbench.

The tasks developed below help you begin learning new areas of the knowledgeware capabilities. It is broken down into two tasks and the instructions required by the user are supplied for each task.

When working in a Japanese environment, remember to check the **Surrounded by the Symbol'** (**Tools->Options->General->Parameters and Measure->Knowledge** tab).

Basic Tasks

This section explains and illustrates how to create various kinds of features. The table below lists the available information.

You can also find useful information in Knowledge Expert Automation Principles (see the CAA documentation). The Knowledge Expert product does not provide you with journaling capabilities, but you can write macros replaying most of the Knowledge Expert operations.

About Rule Bases	• Using a Rule Base stored in a catalog • Importing a Rule Base • Activating and Deactivating a Rule Base • Solving a Rule Base • Storing a Rule Base in a catalog
About Rule Sets	• Interactively Creating a Rule Set • Activating and Deactivating a Rule Set • Displaying the Summary of Errors at the Rule Set Level
About Expert Checks	• Creating an Expert Check • Editing an Expert Check • Activating and Deactivating an Expert Check • Generating a Check Report • Highlighting invalid Features • Performing a Global Analysis of Checks • Customizing Check Reports

About Expert Rules	• Creating an Expert Rule • Editing an Expert Rule
Using Knowledge Expert Tools	• Using the Check Editor • Using the Rule Editor • Using the Object Browser • Using Objects Library • Using the Check Analysis Tool
Using Knowledge Expert Language (KWE)	• Using Types in the Check/Rule Editor • Using Types Attributes • Using the Filter Editor • Using Functions • Declaring Variables • Using Control Structures

About Rule Bases

The Knowledge Expert application allows you to create and manipulate relation-type features. These particular features are organized into a hierarchy. The *rule base* object is located at the top of this hierarchy (see the graphic below.)

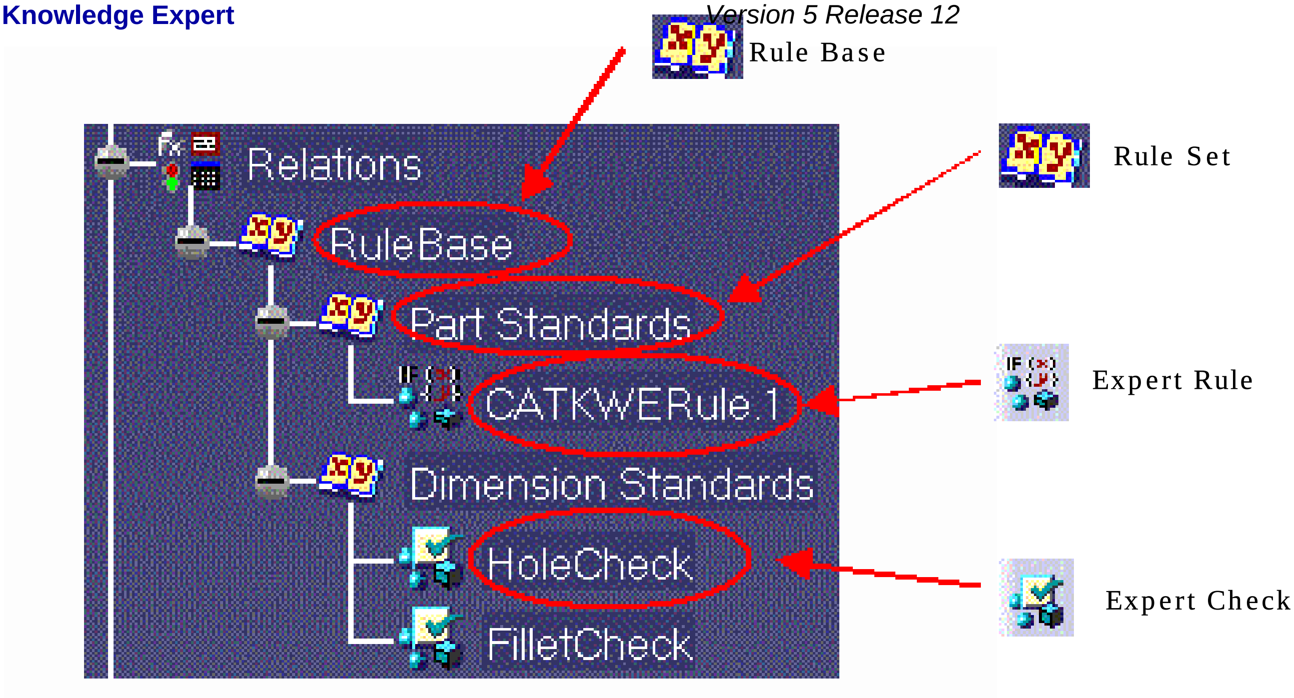
- A rule base is a feature located at the top of the Expert Rule/Check hierarchy (see graphic below).
- A rule base is automatically created when accessing the Knowledge Expert workbench.
- Only one rule base can be added to a CATProduct or a CATPart. But a CATProduct with its rule base can refer to components having their own rule bases.
- A rule base can be activated or deactivated. It can be made up of several rule sets. When a rule base is deactivated, the features below are deactivated too.
- A rule base has no effect on a document until a solve operation is launched. The purpose of the solve operation consists in firing (or executing) the active relations in the active rule sets.

To know more about Rule Bases, see:

- Storing Rule Bases in a Catalog
- Using a Rule Base stored in a catalog
- Importing a Rule Base
- Activating and Deactivating a Rule Base
- Solving a Rule Base

Summary of Tasks

Please find below the Knowledge Expert application hierarchy.


Rule Base
Rule Set
Expert Rule
Expert Check
Relations
RuleBase
Part Standards
CATKWERule.1
Dimension Standards
HoleCheck
FilletCheck

Storing Rule Bases in a Catalog

This task explains how to create a catalog storing a rule base.

1. Open the KwxCatalog.CATPart and the KwxCatalog2.CATPart files.

   Note that the documents you can add to a catalog can be parts or products.

2. From the Start menu, select the **Infrastructure**->Catalog Editor command. The Catalog Editor dialog box is displayed.

3. In the Catalog Editor, select the Chapter1, then click the **Add Family** icon ( ). The Component Family Definition dialog box is displayed.

4. If need be, change the default name of the family (type "Rulebases" for example), and click **OK**.

   In the catalog editor, the new family is added to the tree.

5. Double-click the family, then click the **Add Component** icon ( ), the **Description Definition** dialog box is displayed.

6. Click the **Select external feature** button, select the rule base contained in the KwxCatalog.CATPart specification tree. A catalog containing your rule base is created.

7. .Double-click the family, then click the **Add Component** icon ( ), the Description Definition dialog box is displayed.

8. Click the **Select external feature** button, select the rule base contained in the KwxCatalog2.CATPart specification tree. A catalog containing your rule base is created.

9. Save the created catalog by using the **File->Save(SaveAs)**command then close the Catalog Editor panel. The 2 rule bases have been saved in a file with the .catalog extension.

- Rule bases can be stored in .catalog files in order to be retrieved later on, re-imported or simply applied to any document.
  To know more about this, see Using a Rule Base stored in a catalog or Importing a Rule Base.
- For information on the Catalog, see the *Infrastructure User's Guide*.

Using a Rule Base Stored in a Catalog

Knowledge Expert enables users to use rule bases stored in catalogs and to instantiate them. The User may choose to apply the rule base to the document (**Use only** option), to import it and copy it (**Import** option), and to import it and maintain a link with the original rule base.

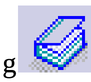
Use Only Option	• Applies the rule base to the document. • Solves the rule automatically.
Import Option	• Copies the rule base into the document as well as its sub-components (rule sets, rules, and checks).
Import with Link Option	• Imports the rulebase as a link and keeps the link with the original rulebase. • Does not copy the content of the rulebase.

Using a Rule Base Stored in a Catalog: Use Only Option
Using a Rule Base Stored in a Catalog: Import Option
Using a Rule Base Stored in a Catalog: Import With Link Option

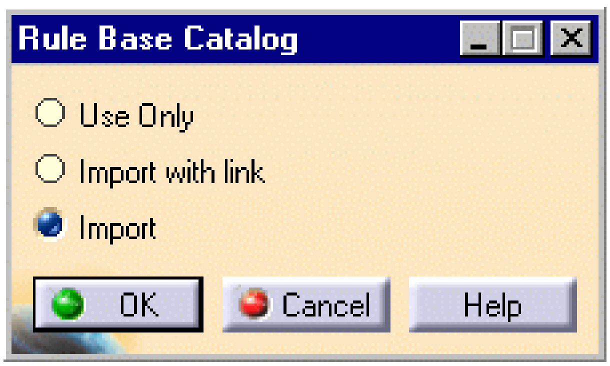
See also the *Creating a customized Toolbar containing a Catalog* topic in the *Infrastructure User's Guide*.

Using a Rule Base Stored in a Catalog: Import Option

This task explains how to retrieve and apply a rule base stored in a catalog to a document by using the Use Only Option.

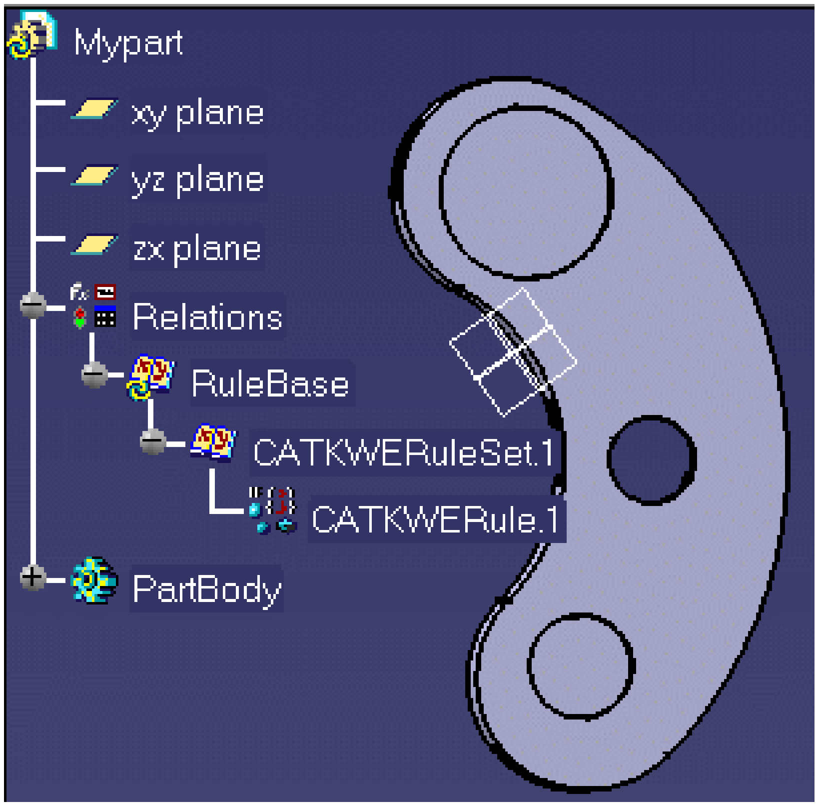
1. Open the document you want to import the rule base to (KwxCatalogImport.CATPart for example).
2. Click the Open catalog  icon. The catalog browser is displayed. Select the KwxCatalogRuleBases.catalog file which contains the rule base to be imported. The Rules family is displayed in the left-hand part of the catalog window.
3. Double-click the RuleBase.

   This dialog box displays.

4. Check **Import** to import the rule base into the document. Click **OK**, then **Close**. The screen below is displayed.

   - The rule base contained in the catalog is imported into the document. In this case, the imported rule base does not have any



link with the rule base of origin and will not be updated if the original rule base is modified.

- The rule base appears in the specification tree under the Relations node.

Formulas cannot be stored in catalogs, only parameters values can be. So, if you have applied a formula to a parameter, only the parameter will be imported together with the rule base.

Using a Rule Base Stored in a Catalog: Use Only Option

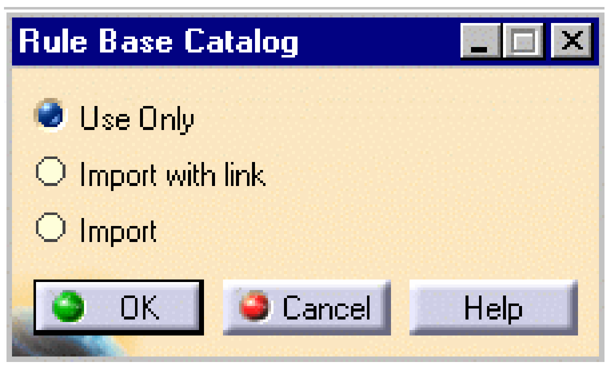
This task explains how to retrieve and apply a rule base stored in a catalog to a document by using the **Use Only** Option.

1. Open the document you want to import the rule base to (KwxCatalogImport.CATPart for example).

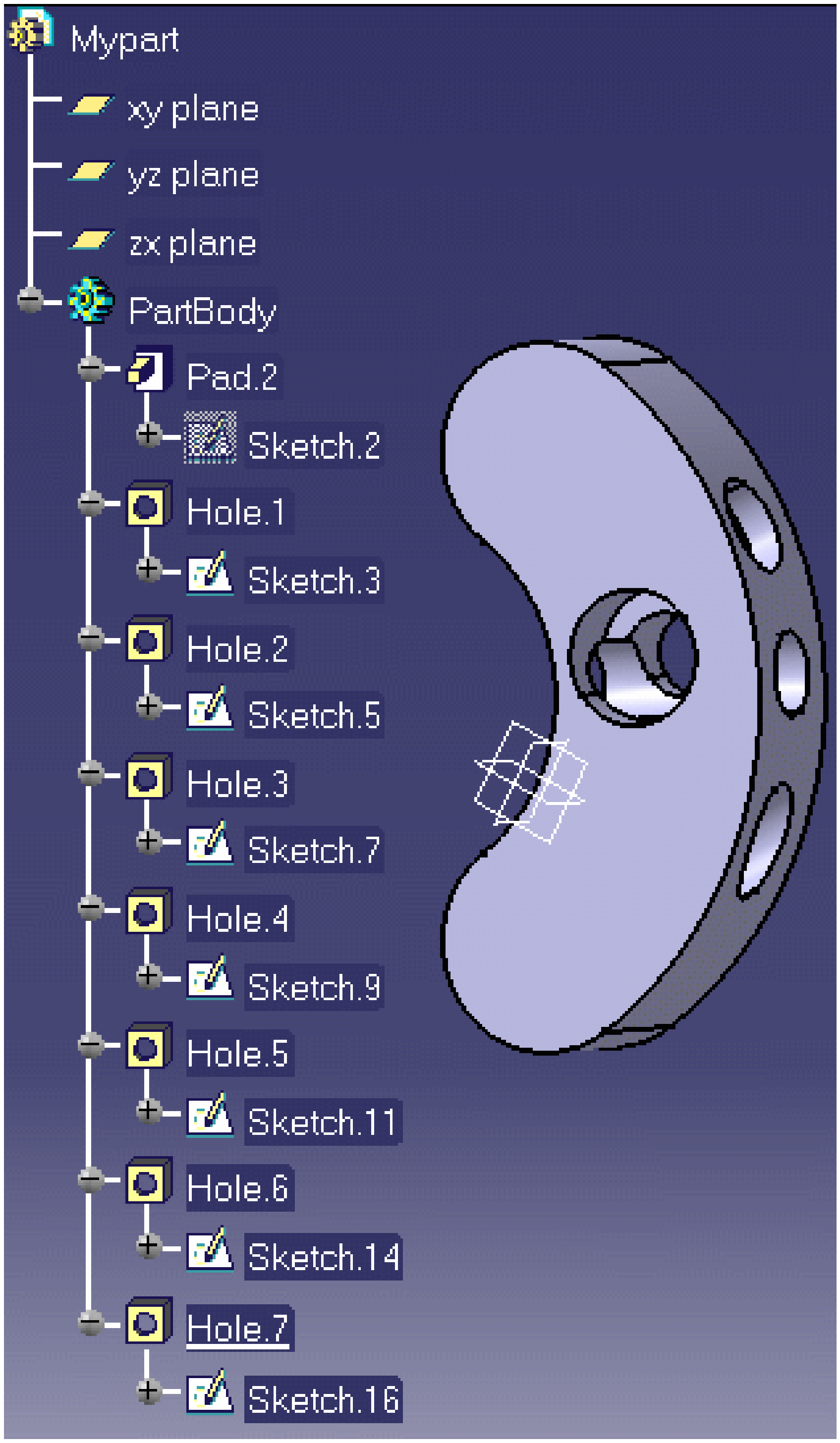
2. Click the Open catalog icon. The catalog browser is displayed. Select the KwxCatalogRuleBases.catalog file which contains the rule base to be imported. The Rules family is displayed in the left-hand part of the editor.

3. Double-click the RuleBase.

This dialog box displays.



4. Check **Use Only** to apply the rule base to the document. Click OK twice, then **Close**.



The rule contained in the imported rule base is applied to Mypart but the rule base does not appear in the specification tree.

If you have applied parameters to the Expert rules or the Expert checks contained in the Rule Base, they will not be imported, and you will have to re-create them.

Using a Rule Base Stored in a Catalog: Import with Link Option

This task explains how to import a rulebase while maintaining the link with the original rulebase.
This scenario is divided into 3 parts:

- you import the rule base
- you modify the original rule base and store the result of the solve operation
- you synchronize the rulebase (contained in the KwxCatalogImport.CATPart file) with the original rulebase (contained in the KwxCatalog.CATPart file).

The import of knowledge rulebases consists in duplicating a rulebase i.e creating an interactive link between 2 rulebases stored in 2 different files. It implies significant model size reduction, and ensures the use of updated knowledge relations.

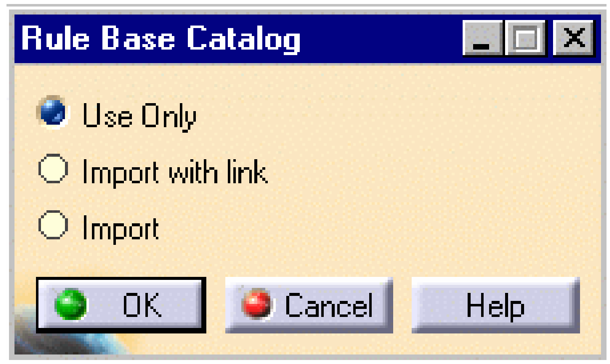
The user can:

- see the structure of the rulebase and navigate through it
- see if the rulebase is updated or not (thanks to an icon)
- see if the rulebase is synchronized or not
- launch corrective actions, and generate reports
- modify settings information (automatic update)

1. Open the KwxCatalog.CATPart file and create a catalog containing the reference rulebase. To know more about catalogs, see the *Infrastructure User's Guide*. Close the Catalog Editor and the KwxCatalog.CATPart file.
2. Create a new part containing holes or open the KwxCatalogImport.CATPart file.
3. Click the **Open catalog** icon ( ). The catalog browser is displayed. Select the catalog you have just created. The family you created is displayed in the left-hand part of the editor.

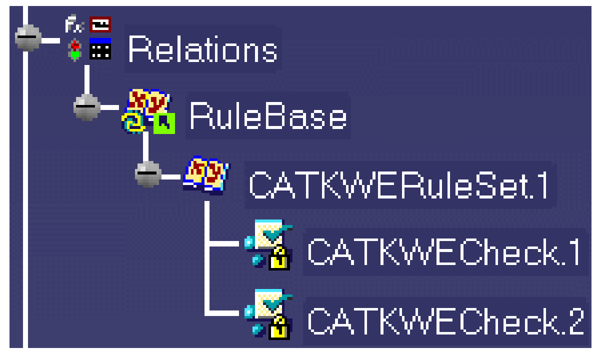
4. Double-click the RuleBase family.

This dialog box displays.



5. Check **Import with link** to import the rule base into the document. Click OK, and **Close**.

The rule base contained in the catalog is imported (with the necessary information only) into the document.



The rule base appears in the specification tree under the Relations node as well as the rules and checks nested into the database.

The  symbol indicates that the link with the original rule base is maintained.

The  symbol indicates that the checks are locked and cannot be modified.

6. Click the Solve icon () to solve the imported rule base.

7. Open the KwxCatalog.CATPart file and modify one of the checks. To do so, proceed as follows:

   - Double-click CATKWECheck.1: the Check Editor opens.

- In the **Condition** tab change the value to 15:

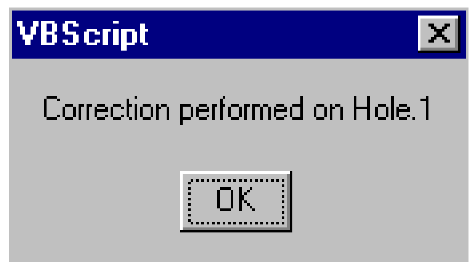
  H.Diameter == 15mm

- Click the **Correction** tab, select **VB Script** in the scrolling list, enter the following correct function and click **OK**:

```
Dim aHole as Hole
Set aHole = H.parent.Item(H.Name)
Dim diam As Length
Set diam = aHole.Diameter
diam.Value = 16
MsgBox("Correction performed on "&H.Name)
```

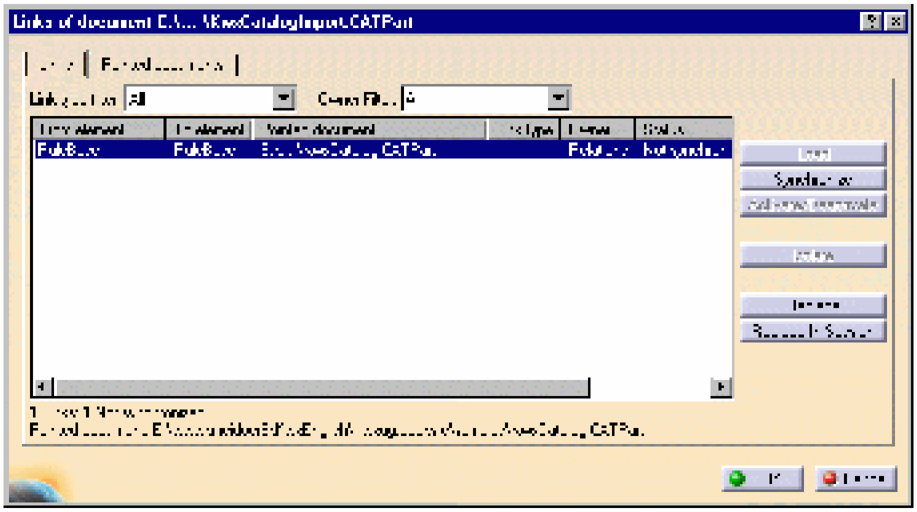
- Right-click the rulebase and select **RuleBase object->Manual Complete Solve** from the contextual menu to solve the rulebase.

- Right-click the CATKWECheck.1 check and select **CATKWECheck.1 object->Correct Function** from the contextual menu to launch the correct function. The following message displays for every corrected hole (see opposite).



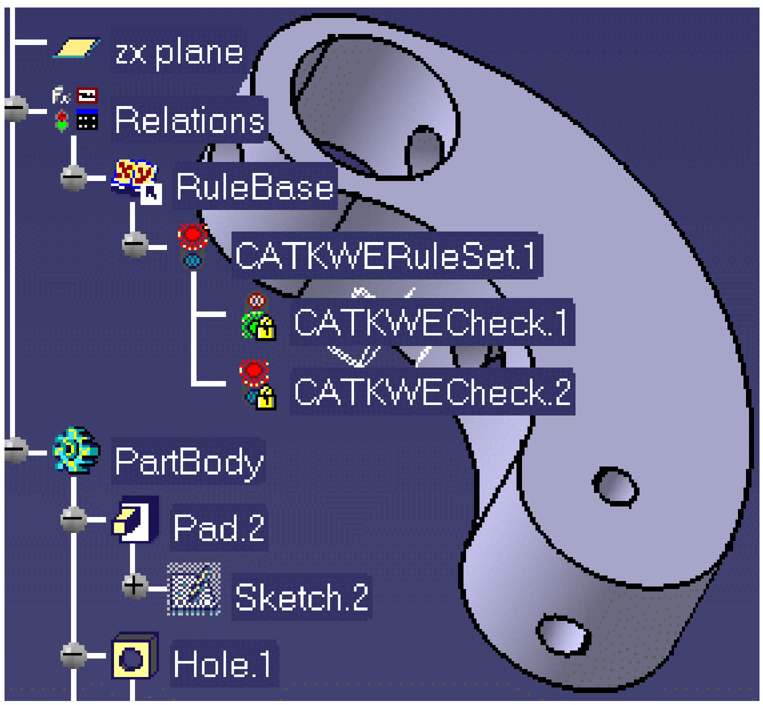
- Save the file and close it.

8. In the KwxCatalogImport.CATPart file, select the **Edit->Links** command. The **Links of document** window opens (see below).

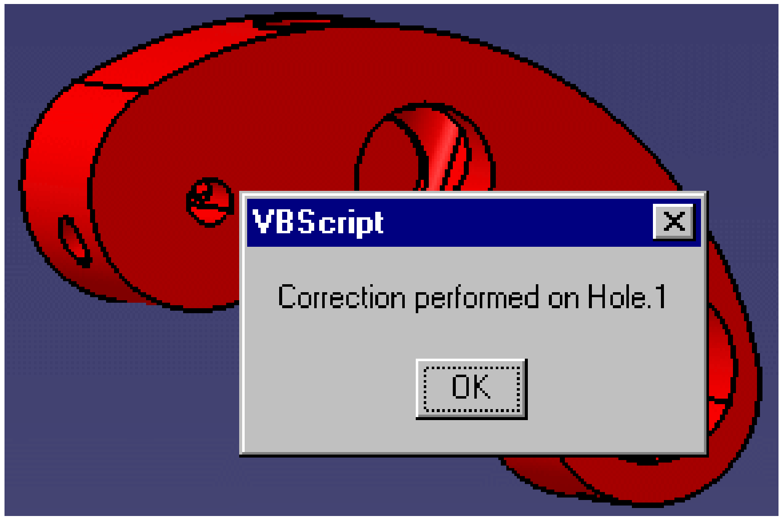


Click the graphic opposite to enlarge it.

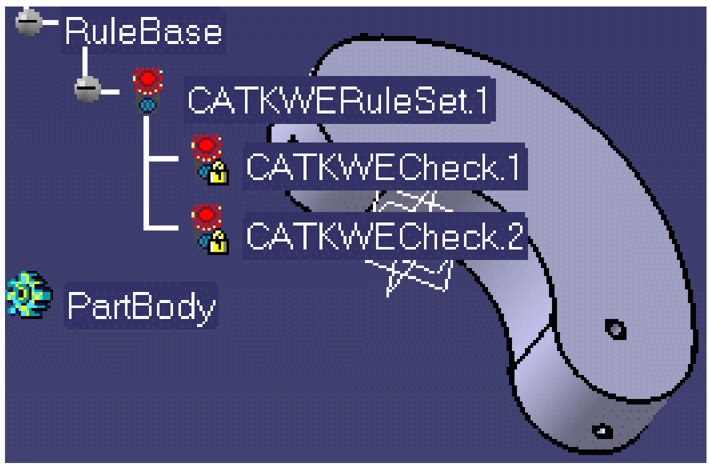
9. Click the Synchronize button and click **OK**. The rulebase is synchronized.



10. Right-click the CATKWECheck.1 and select the **Correct Function** command. The correct function is launched.



11. Click the Solve icon: the document is updated (see below.)



Importing a Rule Base

This task explains how to import a rule base from an external file. The imported file must be a .CATProduct file while the receiving file can be either a .CATPart or a .CATProduct. The rule base is the only feature imported from the external document.

The imported rule base should be located right below the root product. A rule base located below one of the product components will not be imported.

1. Open the document you want to import the rule base to (KwxCatalogImport.CATPart for example).
2. Access the Knowledge Expert workbench.
3. Click the icon. A file selection box is displayed.
4. Select the KwxInportRuleBase.CATProduct file.
5. Click Open to import the rule base from the external file you have just selected. The expert rules/checks are added to the specification tree.

- The document the user can import a rule base to can be a .CATProduct or a .CATPart
- Only rule bases contained in CATProduct files can be imported (this step is meaningful if the selected file contains a non-empty rule base).
- If a Rule set is imported whose name is identical to the name of an existing Rule set contained in the document, a panel opens asking you whether you want to replace it.

Activating and Deactivating a Rule Base

This task explains how to activate and deactivate a Rule Base.

Prior to starting this task, make sure you have created or imported a Rule Base from an external document.

To activate and deactivate a Rule Base, proceed as follows:

- In the specification tree, right-click the rule base object, then select the **rulebase object ->Inactivate** command from the contextual menu.

Solving a Rule Base

This task explains how to solve a rule base.

The rule base must have been created or imported from an external file, and its parameters set.

To solve a rule base, proceed as follows:

- Click the icon in the Knowledge Expert Workbench. This icon is activated when the document or one of the rule base relations has just been modified. A to-be-solved rule base is displayed in the specification tree with a solve icon. Click the Solve icon in the workbench to solve the rule base.

  -or-

- Right-click the rule base and select the RuleBase Object-> Manual Complete Solve command from the rule base contextual menu (Initialization). This command solves the rule base no matter what its status is (to-be-solved or not).

  -or-

- Right-click the rule base and select the RuleBase Object->Manual Optimized Solve command from the rule base contextual menu to perform a solve if changes have been made since the last solve operation was performed.

About Rule Sets

The Knowledge Expert product allows you to create and manipulate relation-type features. These particular features are organized into a hierarchy. The *rule sets* gather rules, checks, and other rule sets (see graphic below.)

- A Rule Set is a feature that gathers rules and checks. There can be several rule sets in a rule base. The purpose of a rule set is to gather the relations which have something in common, process the same kind of features or are meaningful only when used together.

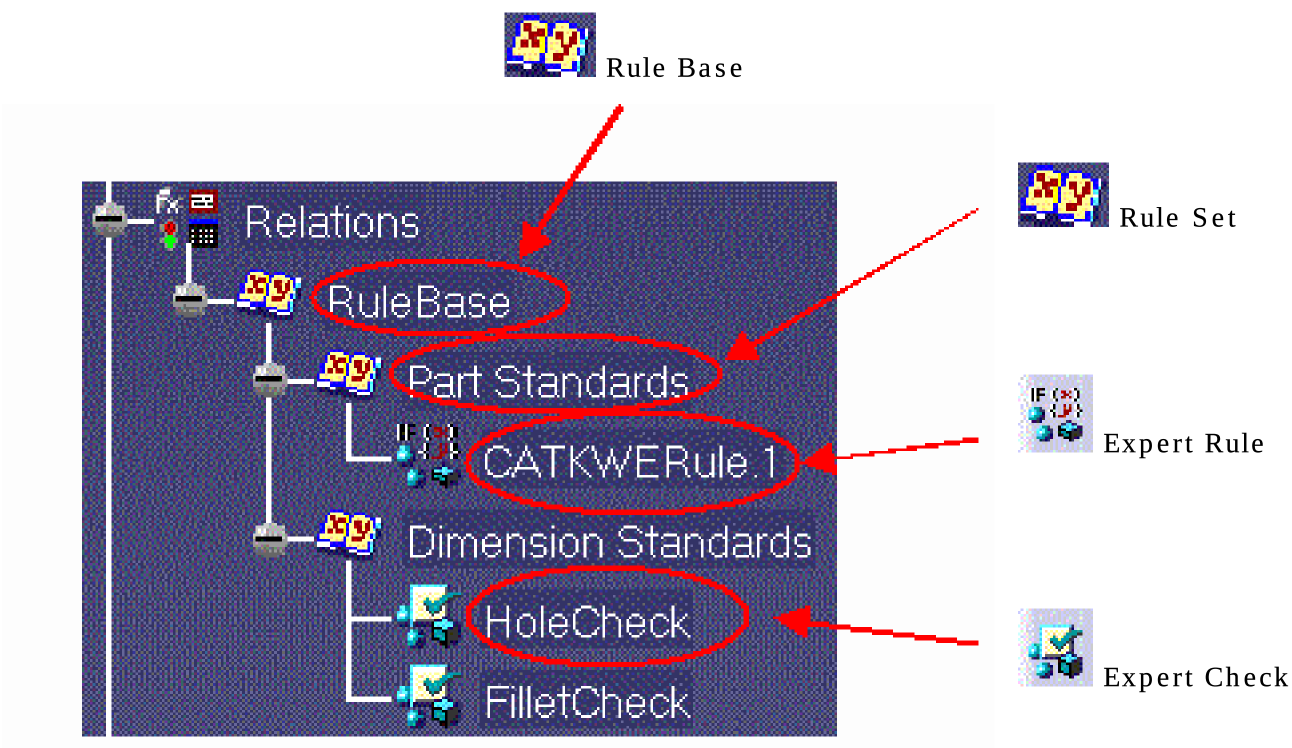
- Rule sets are either automatically created upon creation of rules and checks or interactively created. Rule sets can be nested within a rule base.

To know more about Rule Sets, see:

- Interactively creating Rule Sets
- Activating and Deactivating a Rule Set
- Displaying the Summary of Errors at the Rule Set Level

Summary of Tasks

Please find below the Knowledge Expert feature hierarchy.



Creating Rule Sets interactively

P2

This task explains how to add rule sets to rule bases or to rule sets by using the icon.

This function was developed to enable the user to add rule sets under rule bases and thus create a rule sets hierarchy. Rule sets can now be defined and managed to logically structure the corporate knowledge base: the user can classify the rules and checks he created by process, for example.

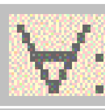
For the Relations node to be correctly displayed in the specification tree, make sure Relations is checked in the Options dialog box (**Tools->Options...->Infrastructure->Part Infrastructure->Display**)

1. Open the KwxRuleSet1.CATPart.
2. Select the root of the specification tree, and from the Start menu, select **Knowledgeware->Knowledge Expert**: a rule base is automatically added to the Relations node.
3. Select the rule base and click the **Rule set** icon. If need be, change the name of the rule set (RuleSet1 in this scenario) and click OK.
4. Click the **Expert Rule** Icon, change the name of the rule (ExpertRule in this scenario) and click **OK**. The Rule Editor opens.
5. Enter the following script in the Editor, then click **Apply**, and **OK**.

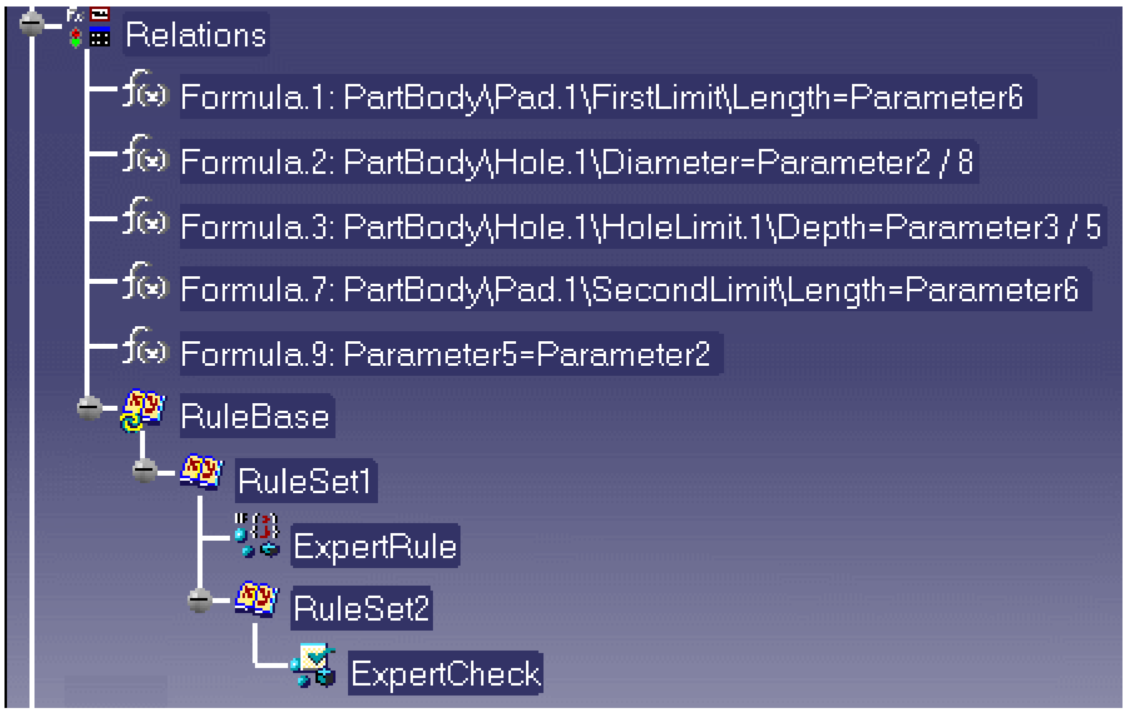
∀:	**H:Hole**
	if(H.HoleType == "Simple") { H.Diameter = 24mm }

6. Select the rule set and click the **Rule set** icon (). If need be, change the name of the rule set (RuleSet2 in this scenario) and click **OK**.
7. Select the Rule Set you have just created (RuleSet2), click the **Expert Check** icon, change the name of the Check (ExpertCheck in this scenario) and click **OK**. The Check Editor opens.

8. Enter the following script in the Editor, then click **Apply**, and **OK**.

	**H:Hole**
	H.Diameter > 10mm

9. Click **Apply**, and **OK**. The Relations node of the specification tree now looks like the one below:



- The user can nest as many Rule Sets as required into a Rule Base.
- The user can nest as many Rule Sets as required into another Rule Set.
- The user can create as many Expert Rules or Expert Checks as required under each Rule Set.

- This function is especially helpful when activating and deactivating rule sets. To know more, see Activating and Deactivating a Rule Set.

Activating and Deactivating a Rule Set

This task explains how to activate and deactivate a Rule Set.

To activate and deactivate a Rule Set, proceed as follows:

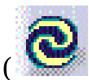
- In the specification tree, right-click the rule set to be activated or deactivated, then select the **ruleset object->(De)activate** command from the contextual menu.

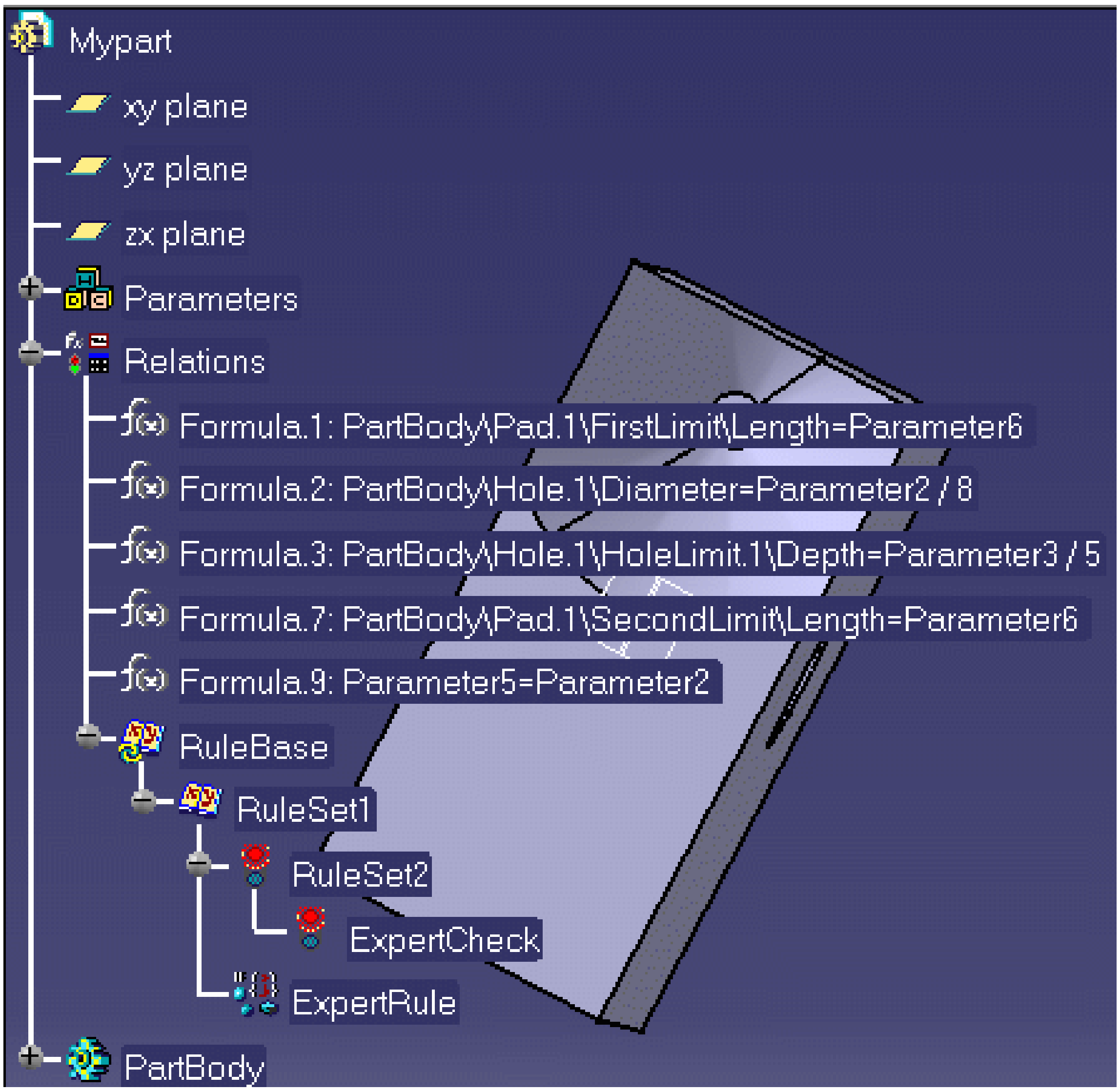
Displaying the Summary of Errors at the Rule Set Level

This task explains how a rule set appears in the specification tree when one of the checks (or all the checks) it contains fails.

This summary displayed at the rule set level is intended to improve and simplify the management of rule sets.

1. Replay the scenario described in Interactively Creating a Rule Set.

2. Click the Solve icon ().



The check is invalid and a red light is displayed at the Check and at the Rule Set levels.

About Expert Checks

The Knowledge Expert product allows you to create and manipulate relation-type features. These particular features are organized into a hierarchy. The *rule base* object is at the top of this hierarchy, the *expert rules* and *expert checks* are the terminal objects. In between you can find the *rule sets* which gather rules and checks (see the graphic below).

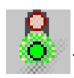
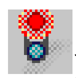
- An *Expert check* is a relation which only checks that a condition is true for the objects of one or more given types. They do not modify the document they are applied to.

An expert check is made up of two parts:

1. The definition of the feature types the check applies to:
   H:Hole
2. The check body:
   H.Activity == true

The check above tests the activity of the features of Hole type belonging to your document. An expert check is valid (the condition specified is fulfilled for all the objects) or invalid (the condition is not fulfilled for all the objects).

The list of objects and attributes to be used in expert rules and checks is displayed in the object browser. See Using the Object Browser.

The icons in the specification tree turn to green() or red() depending on whether the checks are valid or invalid. A check which is partially valid is red. When a check is invalid, you can find out what features are valid or invalid by generating and editing a report. If need be, you can also specify a correction method.

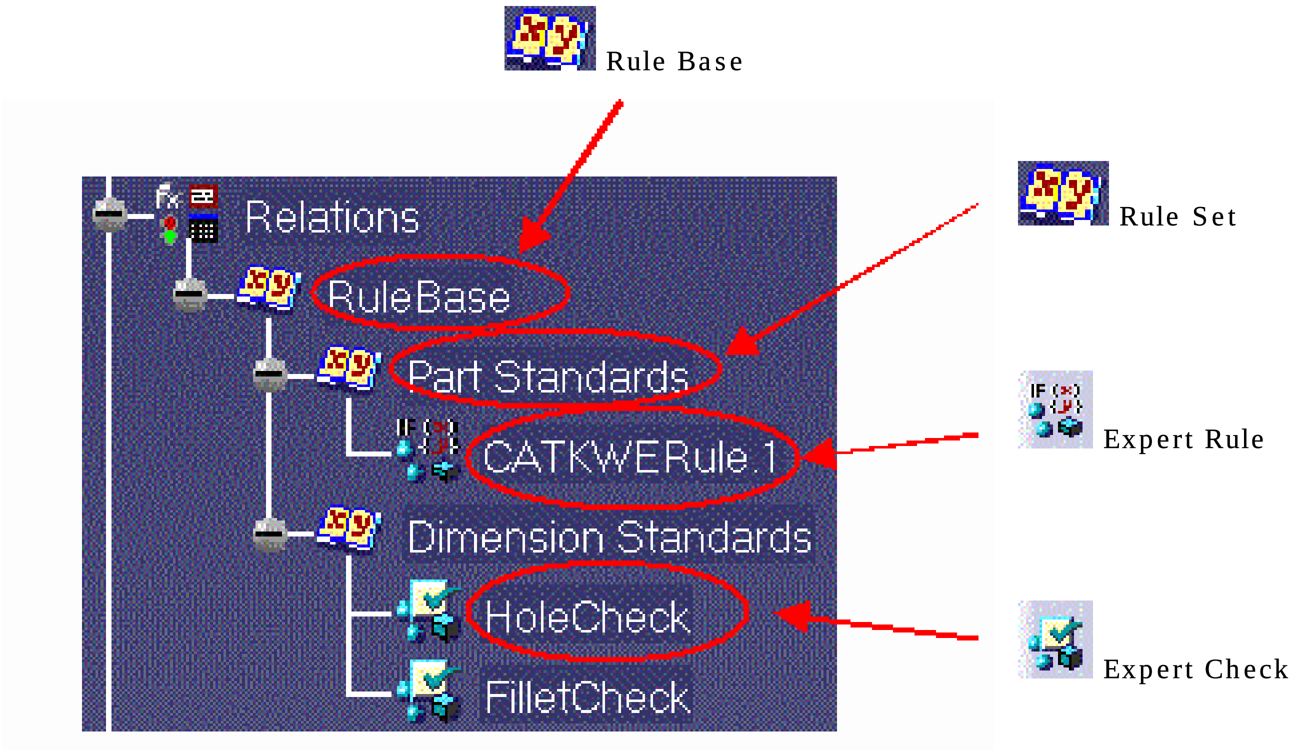
For more information on the expert rule/check syntax, see the Using the Knowledge Expert tools.

To know more about Expert Checks, see:

- Creating an Expert Check
- Editing an Expert Check
- Activating and Deactivating an Expert Check
- Generating a Report
- Customizing Reports
- Highlighting Invalid Features
- Accessing the Check in the Check Body
- Performing a Global Analysis of Checks

Summary of Tasks

Please find below the Knowledge Expert feature hierarchy.



Creating an Expert Check

This task explains how to create a check which detects whether all the holes are activated and have a 11mm diameter.

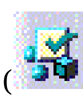
Prior to performing this task, make sure you have selected the required packages. To load the required libraries, proceed as follows:

1. Select the **Tools->Options** command to open the Options window, then select **General->Parameters and Measure**, and click the Knowledge tab.
2. In the **Parameter Tree View** area of the **Knowledge** tab, check the **With value** and **With Formula** options.
3. Click the **Language** tab and check the **Load extended language libraries** option and select the libraries you want to load (**PartDesign** in this scenario).

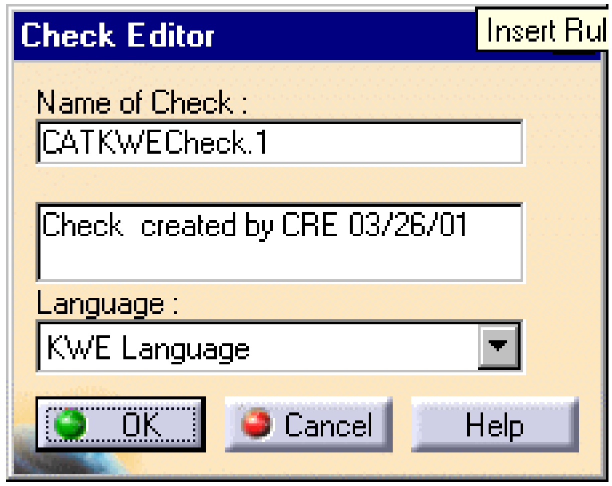
1. (Re-)access the Knowledge Expert workbench.

   a. Select the root item in the specification tree.

   b. In the **Start** menu, select **Knowledgeware-> Knowledge Expert** workbench.

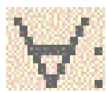
2. Click the Expert Check icon ().

3. Select the RuleBase relation in the specification tree. The following dialog box is displayed.



4. If need be, replace the default name and description for the check to be created.

   Select the KWE Language, then click OK. The expert check editor is displayed.

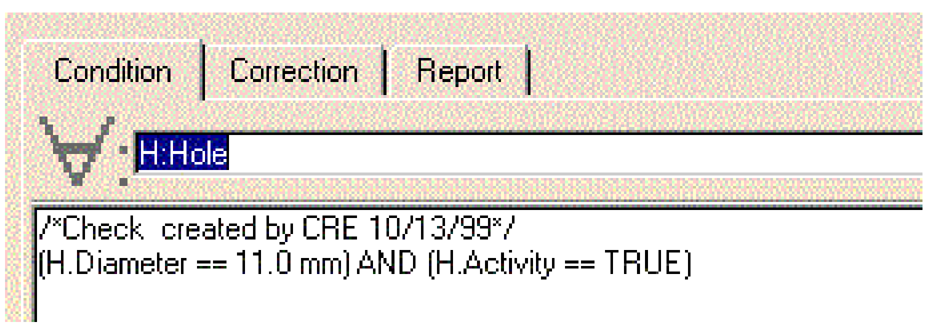
5. Use the area with the  symbol to specify the feature type you want to apply the expert rule. The following syntax should be applied:

   H:Hole

6. Copy/Paste the code below from your browser to the edition box:

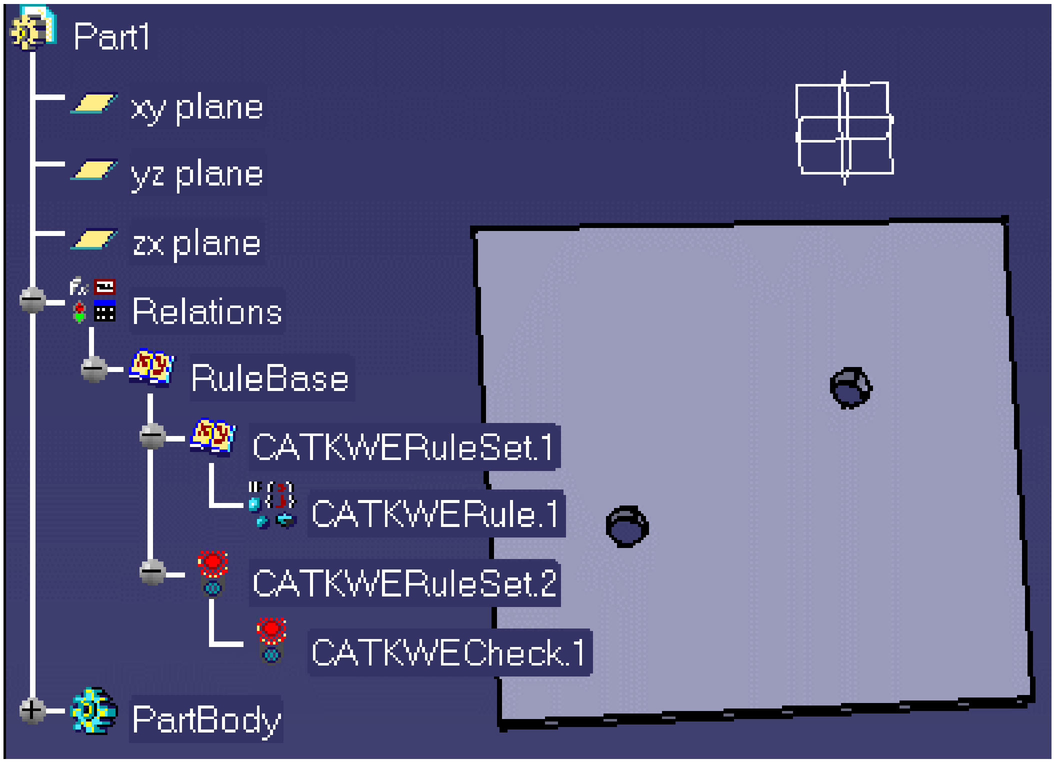
   (H.Diameter == 11.0 mm) AND (H.Activity == TRUE)

   The check editor now looks something like this:



7. Click **OK**. A check is added to the rule base in the specification tree.

8. Click the  icon to solve the rule base. Your document looks something like this:



The light icon associated with the check has turned to red, indicating that the check is not valid (all the holes have a diameter of 10.0mm).

Right-click the check in the specification tree, and select the **Highlight Failed Component** command. This highlights the features that don't fulfill the criteria specified in the check.

Editing an Expert Check

This task explains how to edit an Expert Check.

An Expert check must have been created. For more information on how to create an Expert Check, see Creating an Expert Check.

To edit an Expert Check, proceed as follows:

- In the specification tree, double-click the check to be edited, then modify its statements in the Check Editor.

-or-

- In the specification tree, right-click the rule to be edited, then select the **Check Object->Definition** command from the contextual menu.

For more information on the Object Browser available from the Check Editor, see Using the Browser.

Activating and Deactivating an Expert Check

This task explains how to activate and/or deactivate an Expert Check.

- In the specification tree, right-click the check to be activated/deactivated, then select the **Activate/Inactivate** command from the contextual menu.

Accessing the Expert Check in the Check Body

The task described below explains how to access the check itself in the check body by using the "Thischeck" variable.

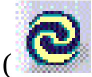
"Thischeck" and "Thisrule" are variables created to help the user write rules and checks. These 2 variables enable the user to automatically reference the check or the rule he is working with. It enables him to:

- Access the parameters located below the rule or the check (see scenario below),
- Compare various elements.

1. Create a pad containing holes or open the KwxThisCheck.CATPart file.
2. Access the Knowledge Expert workbench and click the **Expert Check** icon ( ).
3. Change the name of the Check (ExpertCheck in this scenario), select the KWE language (by default) and click **OK**. The Check Editor opens.
4. Set the new parameter of type to **Length**, then click the **New Parameter of type** button, set the length value to 15 mm, and click OK.
5. In the Check Editor, enter the script indicated in the column "**With the ThisCheck method**" (see table below). Click **OK**.

This script enables the user to check that the diameters of the holes contained in this CATPart file are superior to 15 mm.

With the ThisCheck method	Without the ThisCheck method
For all field: P:Hole P.Diameter > ThisCheck->GetAttributeReal ("LENGTH.1")	For all field: P:Hole ; C1:KWECheck C1.Name == "CATKWECheck.1" /*Indicate the name of the check*/ => P.Diameter > C1->GetAttributeReal ("LENGTH.1")

6. Click the **Solve** icon (). The rule set lights turn to green indicating that the check could be run correctly.

7. Click here to display the result of the scenario.

Highlighting Invalid Features

This task explains how to highlight invalid features after a check has been performed.

Prior to performing this task, make sure a check has been created, and the rule base has been solved.

1. In the specification tree, right-click the check.
2. Select the **Highlight Failed Components** command from the contextual menu.

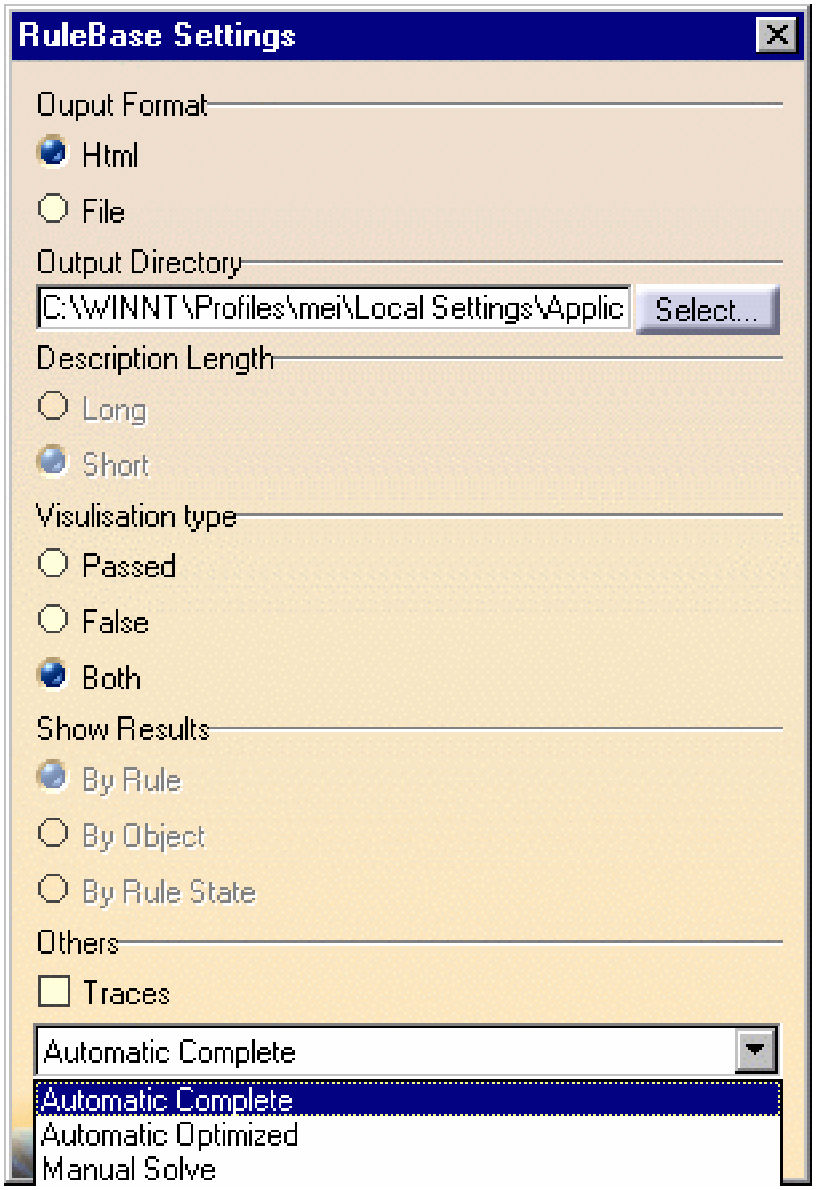
Generating a Check Report

This task explains how to generate a check report after a solve operation has been carried out, to define the rule base settings and to apply a correction function to the check.
The data logged in the generated report as well as the report format depend on the rule base settings.

Prior to carrying out this task, you must have completed Launching a Check Correction Method.

Note that the generated check report will only be based on the selected rulebase. To generate a report based on all the checks of the document, see Performing a Global Analysis of Checks.

1. Expand the specification tree, right-click the rule base object under Relations, and select the **Rulebase Object->Settings** command from the contextual menu. The RuleBase Settings dialog box opens:



2. Refer to what follows to fill in the areas:

a. In the **Output Format** area of the window, check:

- **Html** to generate the report in html format.
- File to generate the report in text format. In this mode, the **Description Length** and the **Show results** options are activated by default.

b. In the **Description Length** area of the window, check:

- **Long** to insert the Help message specified at the check creation.
- **Short** if you do not need the Help message.

c. In the **Visualization** area of the window, check:

- **Passed**: to include in the report only information about the features for which the checks are valid
- **False**: to include in the report only information about the features for which the checks are invalid
- **Both**: to include in the report information about all the features on which a check has been applied.

d. In the Show Results area of the window, check:

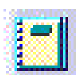
- **By Rule** to organize your report data by rule in the file.
- **By Object** to organize your report data by object.

- **By Rule** State to organize your report data by rule state.

e. In the Others area of the window, check and/or select:

- **Traces** to display the steps of the solve process.
- **Automatic Complete** to perform an initialization and a solve operation on the objects whenever the part is updated.
- **Automatic Optimized** to perform a new solve on the last changes.
- **Manual Solve** to perform a manual solve.

3. Click **OK** to apply the settings to the rule base.

4. Click the  icon to find out what holes are not activated. The html page displayed provides you with the ratio of the holes that fulfill the check. 75% of the holes are activated - Click the check name hyperlink to obtain details on the features satisfying or not satisfying the check. Note the Help message which is displayed in column 5 of the check report.

5. In the specification tree, right-click the check, select the Correct Function command from the contextual menu and perform a Solve. The holes are activated and the part is updated. In the specification tree, the check icon is now green.

Unless you want to modify the check report characteristics, you do not have to re-specify the rule base settings each time you generate a report.

Performing a Global Analysis of Checks

This task explains how to perform an analysis of Knowledge Expert and Knowledge Advisor Checks. The scenario is divided into 2 major steps:

- parameters, formulas and checks are created,
- the checks analysis is run and the checks that failed are corrected.

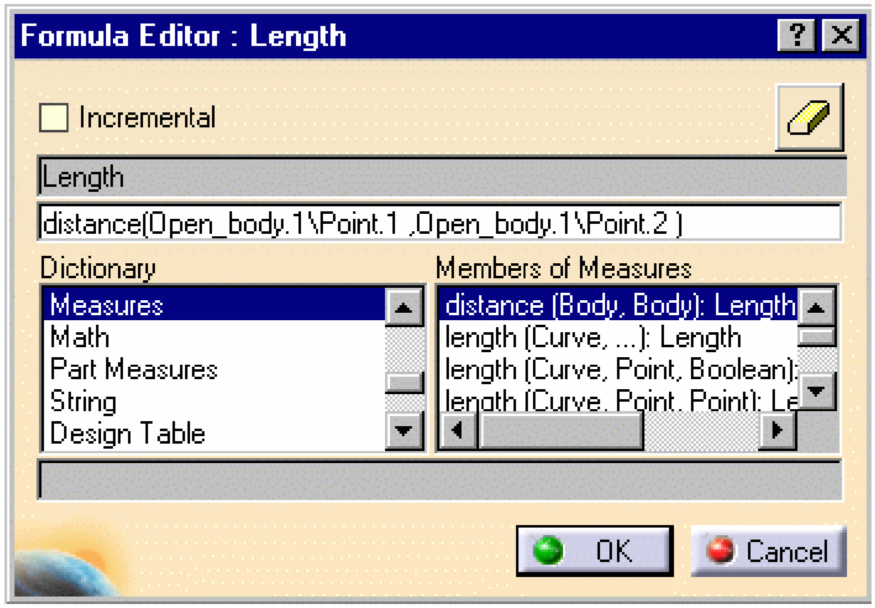
To know more about the Global Analysis tool and the Check Report, see Using the Check Analysis Tool and Customizing Check Reports.

For the check report to be correctly generated, go to **Tools->Options->General->Parameters and Measure ->Report Generation**, and select:

- The Input XSL file under Input XSL. (An XSL file is provided by default. Click here to get a description of the generated XML file.)
- The parameters you want to appear in the report under Report Content.
- The Output directory under Output Directory.

1. Open the KwxCheckAnalysis.CATPart file. From the **Start->Knowledgeware** menu, access the **Knowledge Advisor** workbench.
2. Create a parameter of Length type and assign it a formula. To do so, proceed as follows:
   - Click the icon. The formula editor opens.
   - Select Length in the scrolling list to define the type of the parameter, click the New parameter of type button, change the name of the parameter (Length in this scenario), and click the Add Formula button. The Formula Editor opens.

- Under Dictionary, select Measures, and double-click distance(Body,Body). Position the cursor before the coma and double-click Point.1 in the specification tree or in the geometrical area. Position the cursor after the coma and double-click Point.2 in the specification tree. Click OK, Yes (when prompted for an automatic update), Apply, and OK.



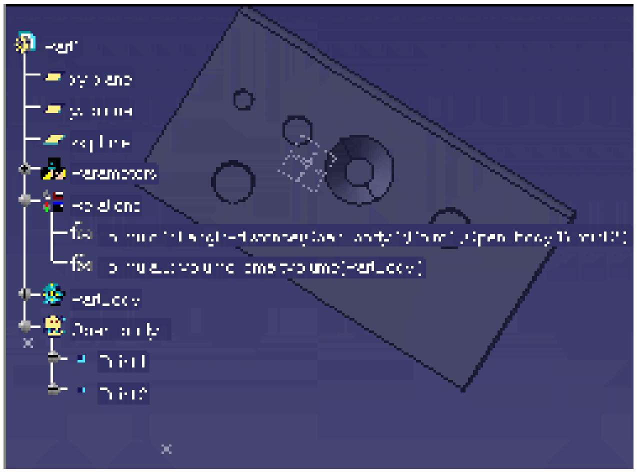
3. Create a parameter of Volume type and assign it a formula. To do so, proceed as follows:

- C

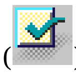
  lick the  icon. The formula editor opens.

- Select Volume in the scrolling list to define the type of the parameter, click the New parameter of type button, change the name of the parameter (Volume in this scenario), and click the Add formula button.

- Under Dictionary, select Part Measures, and double-click smartVolume. Position the cursor between the parentheses and select PartBody in the specification tree. Click OK, Yes (when prompted for an automatic update), Apply, and OK.



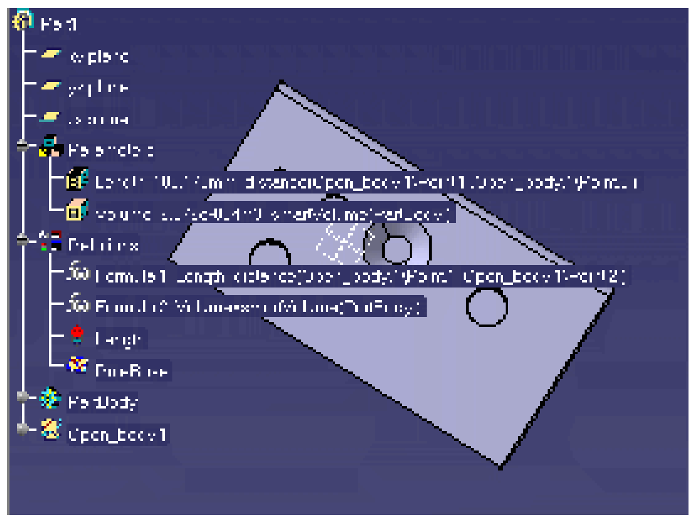
The parameters and the associated formulas are created (click the graphic opposite to enlarge it)

4. Access the Knowledge Advisor workbench, click the Check icon (), change the name of the check (Length in this scenario), and click **OK**. The Check Editor opens.

5. Enter the following script in the editor, then click Apply and OK.

```
Length > 150mm
```

The Knowledge Advisor Check is created (click the graphic opposite to enlarge it).



6. Access the Knowledge Expert workbench, click the Expert Check icon, and change the name of the check (HoleCheck in this scenario). The Expert Rule Editor opens.

7. In the Condition tab, enter the following script:

	H:Hole
Editor	H.Diameter > 15mm

8. Click the Correction tab, select VB Script in the scrolling list and enter the following script in the editor:

```
Dim aHole as Hole
Set aHole = H.parent.Item(H.Name)
Dim diam As Length
Set diam = aHole.Diameter
diam.Value = 16
MsgBox("Correction performed on "&H.Name)
```

9. In the Correction Comment field of the Correction tab, enter the following string, and click **OK**:

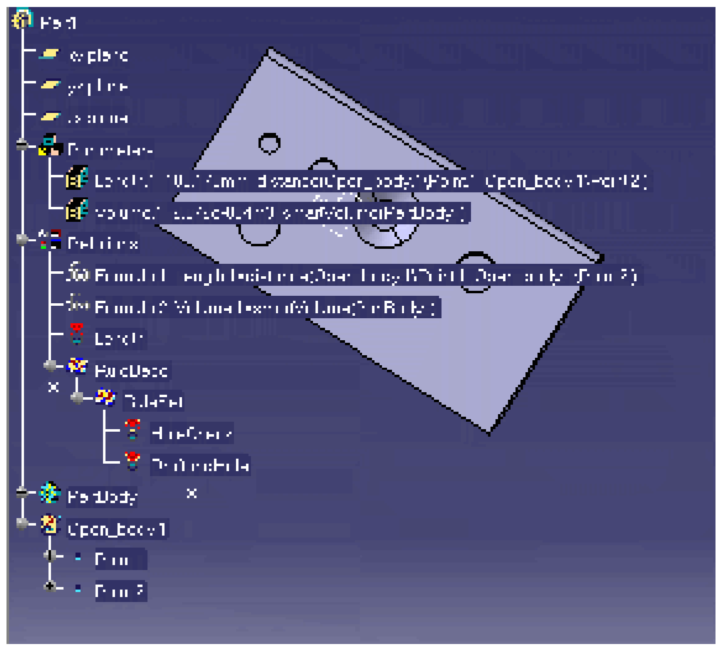
   Holes diameter should be greater than 15mm.

10. Select the Rule Base under the Relations node and click the Expert Check icon, change the name of the check (DraftandHole in this scenario), and click OK. The Expert Check Editor opens.

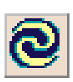
11. In the Condition tab, enter the following script, then click Apply and OK.

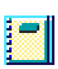
	H:Hole ; D:Draft
Editor	D.Activity AND H.Diameter > 12mm

The checks are created (click the graphic opposite to enlarge it).



12. Click the  icon in the toolbar. The Global Analysis Tool opens.

13. Click the  icon to update the status of the checks. The Checks lights turn to red in the specification tree.

14. Click the  icon. An xml page opens indicating the items that failed. To know more about this report, see Customizing Check Reports.

15. Click the  icon to launch the correction method specified when creating the Expert check (See step 9). The checks have been corrected.

    Only the Advisor check (Length) could not be corrected: The value of the Length parameter is 100.175 mm (as indicated in the report) whereas it should be superior to 150mm (as indicated in the body of the check).

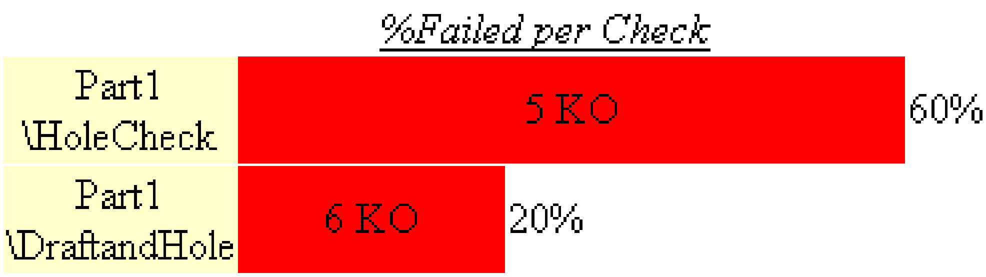
16. To correct the check, modify the value of the Length parameter. To do so, proceed as follows:

    - Double-click Point.1 in the geometrical area. The Point definition window opens.
    - In the H: field, change the value of the point to 150mm. Click **Apply** and **OK**. The light of the check turns to green indicating that the check is passed.

Introducing the Default Check Report

The default check report presents the Expert and Advisor checks that failed.

1. Part1\Length
2. Part1\HoleCheck
3. Part1\DraftandHole



This panel lists the checks that failed and presents a percentage of the failed items per Expert Check.

Advisor Checks report

Check advisor: Part1\Length

Body

```
/*Check created by MEI 11/29/01*/
Length > 150mm
```

This check operates on :

- *Part1\Length = 100.175mm*

The Advisor checks panel lists the Advisor checks that failed and shows the following elements:
- the body of the check (Length>150mm here)
- the item(s) on which the check operates (here, the Length formula).

Expert Checks report

Part1\HoleCheck

Comment					
Input	Part1 \Hole.1	Part1 \Hole.2	Part1 \Hole.3	Part1 \Hole.4	Part1 \Hole.5

	Part1\Hole.1
	Part1\Hole.2
	Part1\Hole.3
	Part1\Hole.4
	Part1\Hole.5

The Advisor checks panel lists the Expert checks that failed and shows the following elements:
- the Input items checked by the check operation.
- the item(s) that failed (here Hole.1, Hole.2, and Hole.3).

Remember that this report should not be used to generate macros or other files. It is provided as information only.

About Expert Rules

The Knowledge Expert product allows you to create and manipulate relation-type features. These particular features are organized into a hierarchy. The *rule base* object is at the top of this hierarchy, the *expert rules* and *expert checks* are the terminal objects. In between you can find the *rule sets* which gather rules and checks (see the graphic below).

The user can use new functions when creating Expert Rules:

- else keyword,
- local variables.

About Expert Rules

- An *expert rule* is a set of instructions whereby you can start an action for any object having a type defined in a type list.
- The action can simply be a message which does not modify the document. More generally it consists of a set of instructions modifying the document. The action specified in a rule can be conditionally executed depending on the value of one or more expressions. In an expert rule, objects are manipulated through their attributes and methods (if any). This applies to checks too.

An expert rule is made up of two parts:

- The (for all)  field in which you define the types list the rule applies to:
  S:Shell ; H:Hole
- The rule body:
  if (S.Activity == true) AND (H.Activity == true)
  Message ("All shells and holes are activated")

To apply an expert rule to a document, you must solve the entire rule base.

To know more about Expert Rules, see:

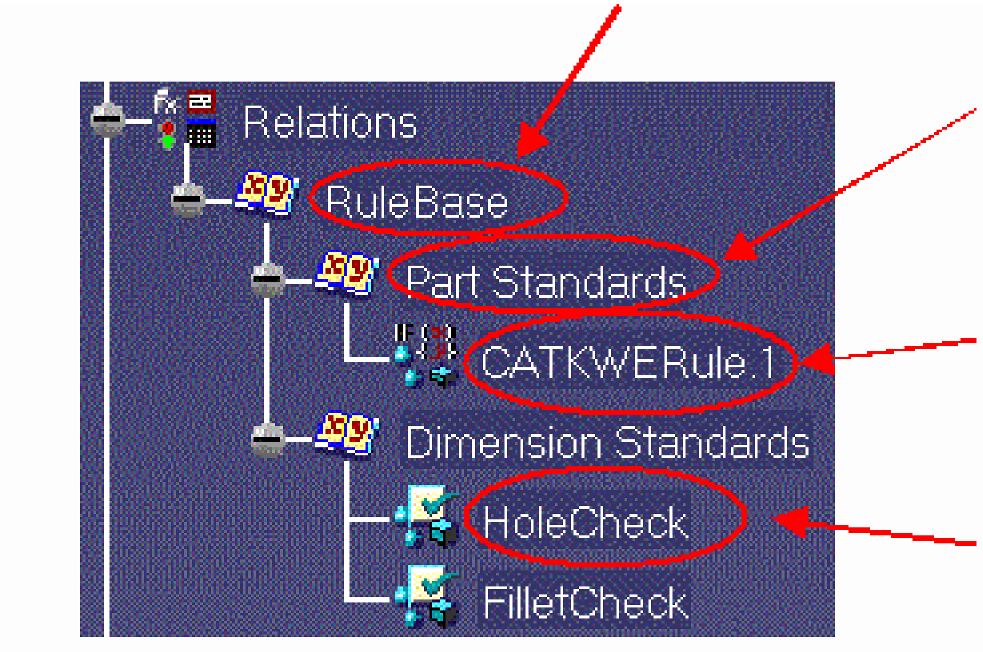
- Creating an Expert Rule
- Editing an Expert Rule

Summary of Tasks

Please find below the Knowledge Expert feature hierarchy.

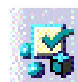
In the figure below, click any of the links to display the related summary of tasks.

 Rule Base tasks



 Rule Set tasks

 Expert Rule tasks

 Expert Check tasks

Creating an Expert Rule

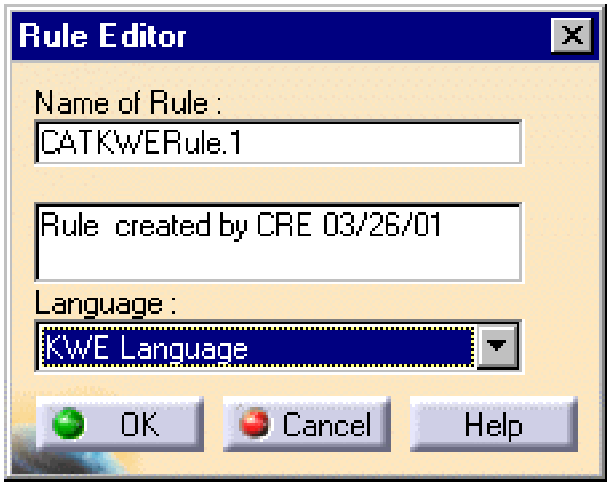
This task explains how to create an Expert Rule.
In the task below, you create a rule which, whenever a hole has a diameter of 50mm, replaces the hole diameter with a 10mm value.

- To carry out this scenario, a basic knowledge of the Part Design product is required.
- The document Update mode must be set to Automatic.

Prior to performing this task, make sure you have selected the required packages. To load the required libraries, proceed as follows:

1. Select the **Tools->Options...** command to open the **Options** window, then select General->**Parameters and Measure**, and click the **Language** tab.
2. In the Language area of the Knowledge tab, check the **Load extended language libraries** option and select the libraries you want to load (here PartDesign).

1. Create a two-hole pad. One hole must have a 50.0 mm diameter, the other a 10.0 mm diameter.
2. Access the Knowledge Expert workbench.
   - Select the root item in the specification tree.
   - In the **Start** menu, select **Knowledgeware-> Knowledge Expert**.
3. Click the **Expert Rule** icon ( ). The following dialog box is displayed.



4. If need be, replace the default name and description for the rule to be created. Select the KWE Language, then click OK. The expert rule editor is displayed.

5. Enter the H:Hole statement in the  area to specify that the rule is to be applied on all the holes and that H will be used as a variable.

6. Copy/Paste the rule below from your browser to the edition box of the editor:

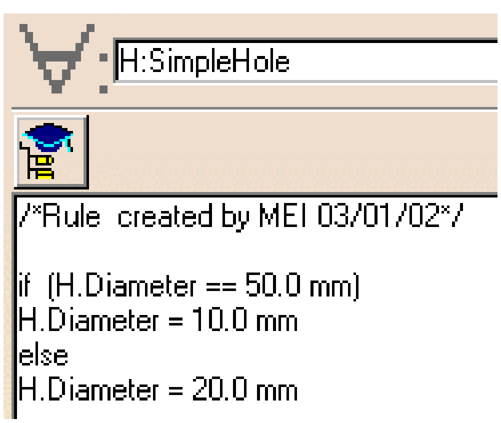
if (H.Diameter == 50.0 mm)

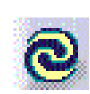
H.Diameter = 10.0 mm

else

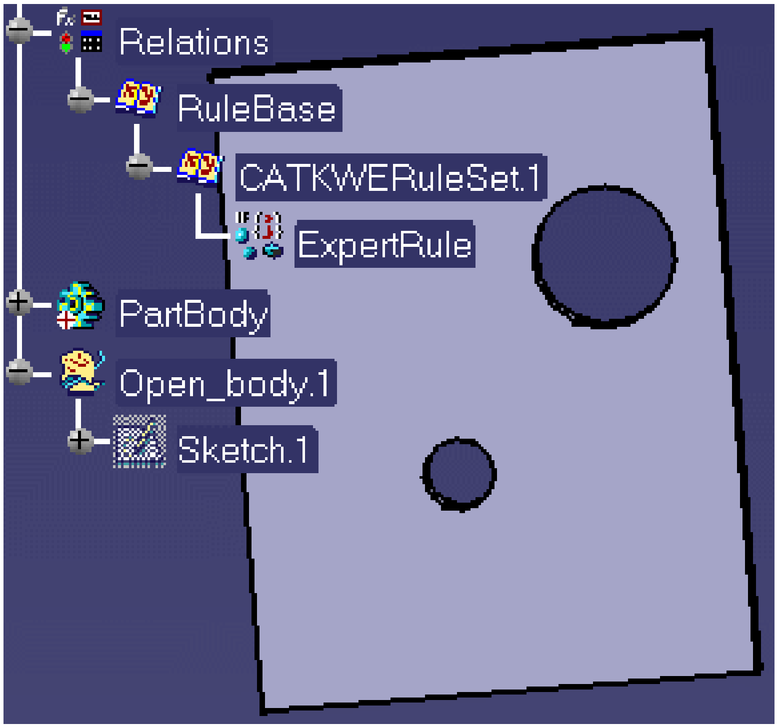
H.Diameter = 20.0 mm

The rule editor now looks something like this:



7. Click **OK**. A rule is added to the rule set in the specification tree. Click the  icon to solve the rule base. If need be, update the document. Here is what you should get

onscreen:



8. Keep your document open and proceed to the next task.

Editing an Expert Rule

P2

This task explains how to edit an Expert Rule.

An Expert rule must have been created. For more information on how to create an Expert Rule, see Creating an Expert Rule.

- In the specification tree, double-click the rule to be edited, then modify its statements in the Rule Editor.

-or-

- In the specification tree, right-click the rule to be edited, and select **Rule.object->Definition** command from the contextual menu, then modify its statements in the Rule Editor.

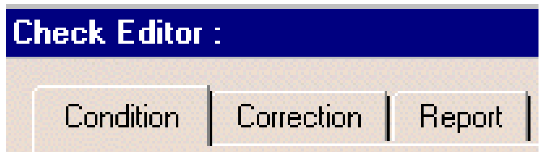
For more information on the Object Browser available from the Rule Editor, see Using the Browser.

Using Knowledge Expert Tools

Using the Check Editor

The Check Editor enables the user to define the check he wants to perform. The Check Editor is made up of three tabs:

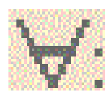
- The **Condition** tab
- The **Correction** tab
- The **Report** tab



Condition tab

The Condition tab is made up of 4 different areas:

Condition area

The "*For all*" field indicated by the universal quantifier  is designed to enable the user to specify the feature types to which the check is intended to apply, to declare the variables names and the variable types.

Example: **H:Thickness**
H:Hole
T:Thickness ; H:Hole ; P:Pad

- Types can be indicated by selecting them in the browser or by clicking the features in the geometrical area or in the specification tree.
- To know more about the syntax used in the Condition area, see the Specifying Feature Types topic.

Check Body area

The check body area is designed for keying in in the check body, which is written in the form of a statement to be checked. In operates on the variables specified in the Condition area (see above).

Example:
H.Diameter == 50.0 mm

The Browser

The browser allows you to access the functions, operators and feature attributes that can be used in an expert check. To know more about the browser, see Using the Object Browser.

Correction tab

The **Correction** tab is made of 2 different areas:

Correction method

The correction method enables you to key in the correction to be applied when the check is not fulfilled (optional).

- **VB Script**: Describes the correction in VB (See Launching a Check Correction Method).
- **Advise Correction**: Displays a comment in the report associated with the rulebase solve or accessed by right-clicking the check in the specification tree and by selecting Correction function
- **URL**: Opens a URL page.
- **User Function**: Describes the correction in KWE language and enables the user to re-use the variables of the condition area in the body of the correction.

Correction comment (available for VB Script only)

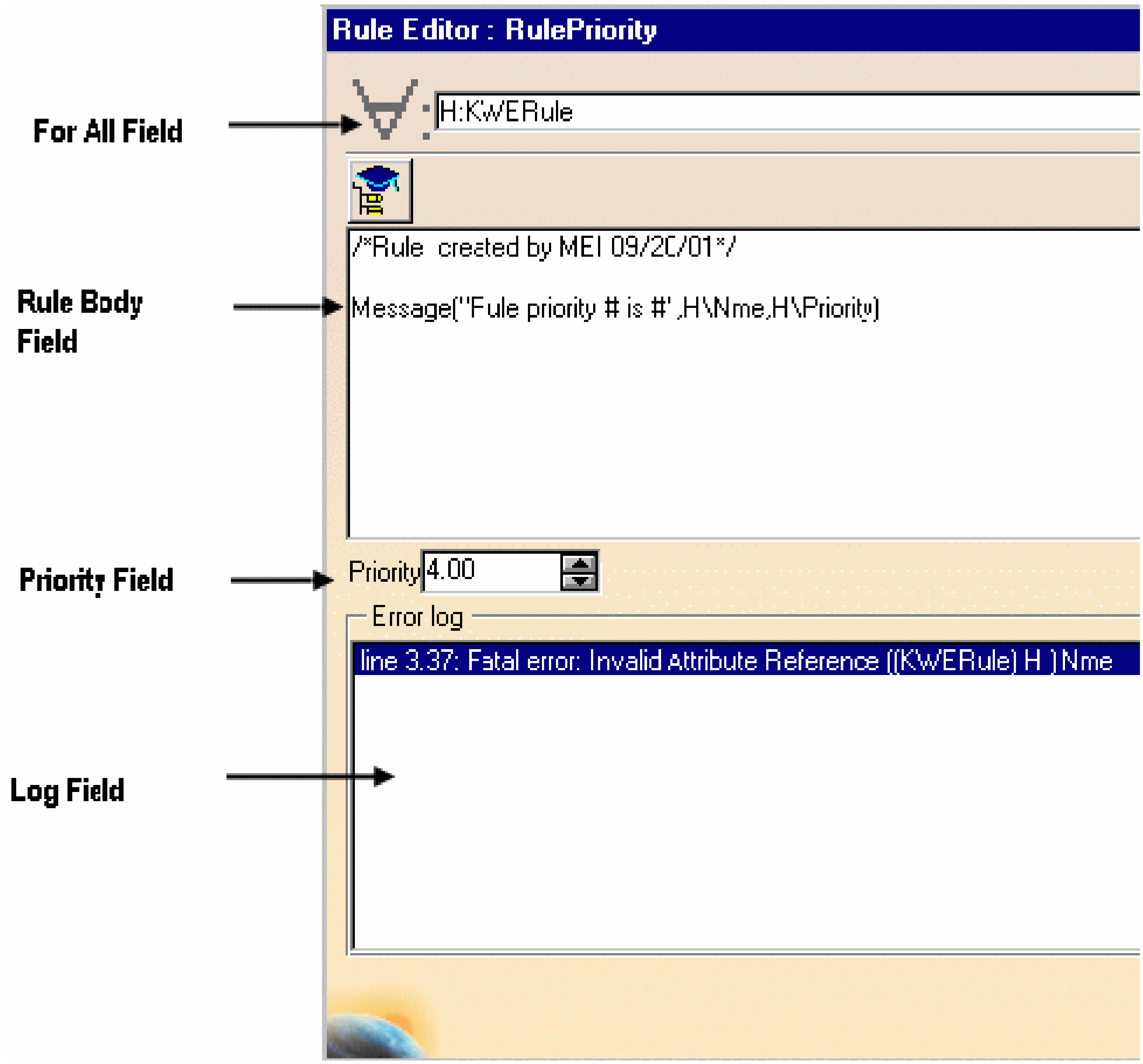
The message you type in the **Correction Comment** area will be displayed in the report generated when you click the Report icon of the workbench. It will also be displayed if you right-click the check in the specification tree and select Correction function

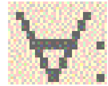
Report tab

The message you type in the edition window of the Report tab will be displayed in the report generated when you click the Report icon of the workbench.

Using the Rule Editor

The Rule Editor enables the user to define the rule he wants to apply. The Rule Editor is made up of 4 different fields:



- The ***"For all"* field** indicated by the universal quantifier  is designed to enable the user to specify the features the rule is intended to apply to, to declare the variables names and their types.

  Example: H:Thickness
  H:Hole
  Tck:Thickness ; Hle:Hole ; P:Pad

To know more about types and their attributes, see Using Types in the Check/Rule Editor and Using Types Attributes.

- The **Rule Body field** is designed to key in the rule body written in the form of a statement to be applied to the variables specified in the *"For all"* field (see above).
- The **Priority field** is designed to enable the user to specify a priority level for the rule.
- The **Error log** field is designed to list the errors appearing in the Rule Body Field. In the example below, "Name" is misspelled, which returns an error in the Error log field.

Using the Object Browser

P2

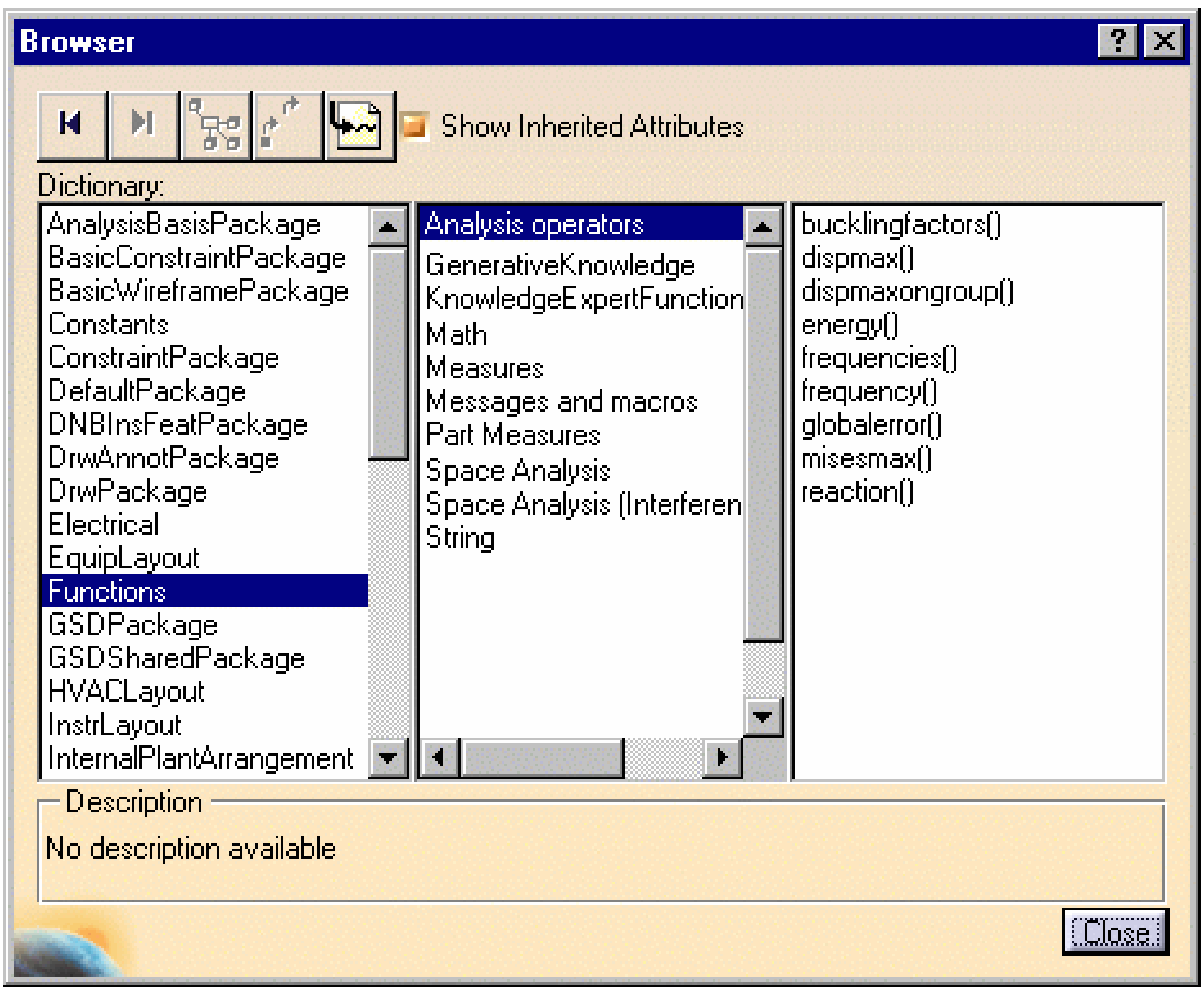
The object browser allows you to access the functions, operators and feature attributes that can be used in expert relations.
It can be accessed from the Rule Editor as well as from the Check Editor.

Packages displayed in the left part of the browser are those you selected from the Tools->Options->General->Parameters and Measure->Language command.

To add or remove packages, proceed as follows:

1. Select the **Tools->Options** command to open the Options window, then select **General->Parameters and Measure**, and click the **Language** tab.
2. In the Language field, check **Load extended language libraries** and select the libraries.

In the rule/check editor, click the  icon to display the Object Browser. The following window opens:



From this window, you can manipulate the list of objects supported by Knowledge Expert through their attributes and methods.

- The left part of the browser displays categories: Applicative packages (MechanicalModeler, PartDesign), units, and constants.
- The central part displays the list of objects belonging to this category (category functions (mathematical functions, launch macro, ...).
- The right part displays the attributes and methods allowing you

to manipulate these objects. The Show inherited attributes check box can be activated here.

Description of the icon bar

The ***Back*** icon.
To return to your last interaction in the browser. Has no action on the rule/check editor.

The ***Forward*** icon.
To go forward to your next interaction in the browser when moving through a series of interactions.

The ***Attribute Type*** icon.
Not to be used in this version.

The ***Parent Feature*** icon.
To retrieve the parent feature as well as its attributes.

Example: Select PartDesign->Shell, then click the Parent Feature icon, the Mechanical Modeler->MechanicalFeature is highlighted, then click again the Parent Feature icon, the Standard->Feature object is highlighted.

The ***Insert*** icon.
To insert the object name in the script.

Functions are now divided into packages (see graphic above). Functions belonging to the Circle Constructors, Direction Constructors, Line Constructors, Point Constructors, Plane Constructors, Surface Constructors, and to the Wireframe Constructors packages have been removed from the browser as they cannot be used in the Knowledge Expert Workbench.

Using the Objects Library

Using the Objects Library

The Object Libraries listed below are those displayed in the Object Browser depending on the packages you selected by using the **Tools->Options...->General->Parameters and Measure->Language->Load Extended Language Libraries** command.

Only highlighted packages are currently documented in the Object Browser.

Automotive BiW Fastening	Optimization
Constants	Manufacturing (See the related manufacturing User's Guides) • MfgActivityPackage • MfgFeatPackage • MfgResourcePackage
Electrical	PartDesign
Equipment Support Structure	PartShared Package
Functions	Product Package
FSAnalysis	Structure Detail Design
FSConstraint	Structure Functional Design
FSGeometry	Structure Preliminary Layout
FSSharedAnalysis	Standard Package
FSSharedGeometry	Topology Package
KnowledgeAdvisor	TPS Package
KnowledgeExpert	

To know more about the packages selection, see the *Infrastructure User's Guide*.

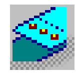
Automotive BiW Fastening Package

Please find below the different exposed types of the Automotive BiW Fastening Application. Click the desired type to access the related page.

BfmJoint
BfmJointElement
BfmBody
BfmSpotPoint

BfmJoint

Description

Describes the BiWJoint feature of BfmJoint type you create when you click the  icon in the Automotive BiW Fastening workbench.

Inheritance path

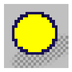
Standard->Feature->Automotive BiW Fastening->BfmJoint

Attribute

BfmFELS
Defines the Forecast Elements Count, which are contained in this BiWJoint.

BfmJointElement

Description

Describes all Fastener features of BfmJointElement type. For example, the Welding BiWSpotPoint that you create when you click the  icon in the Automotive BiW Fastening workbench, is a feature of BfmJointElement type.

Inheritance path

Standard->Feature->Automotive BiW Fastening->BfmJointElement

Attributes

BfmJID
Defines the Joint Name which contains this JointElement

BfmJBID
Defines the Joint Body Name which contains this JointElement.

BfmMID
Defines the Manufacturing Code of this JointElement.

BfmPCATS
Defines the Process Category of this JointElement (Welding, Adhesive, Sealant, BiW Mechanical, Unspecified or other).

BfmPTYPS
Defines the Process Type of this JointElement (for example, 21,14... or other).

BfmREG
Defines the Regulation Attribute of this JointElement (A,B,C,D or other)

BfmROB
Defines the Robustness Attribute of this JointElement (A,B,C or other).

BfmFIN
Defines the Finish Attribute of this JointElement (A,B, C, D or other).

BfmSTY
Defines the Stacking Type of the JointBody which contains this JointElement.

BfmJointBody

Description

Describes the BiWJointBody feature of BfmJointBody type that you create when you click the  icon in the Automotive BiW Fastening workbench.

Inheritance path

Standard->Feature->Automotive BiW Fastening->BfmJointBody

Attributes

BfmFELS
Defines the Forecast Elements Count, which are contained in this BiWJointBody.

BfmSTY
Defines the Stacking Type with which the Contact Zones referenced by this BiWJointBody are assembled.

BfmSpotPoint

Description

Describes the BiWSpotPoint feature of BfmSpotPoint type you create when you click, for example the icon (a Welding SpotPoint is created) or the icon (an adhesive SpotPoint) in the Automotive BiW Fastening workbench.

Inheritance path

Standard->Feature->Automotive BiW Fastening->BfmSpotPoint

Attributes

BfmDIA

Defines the diameter of this JointElement (this attribute type is length and its unit is mm).

BfmMAT

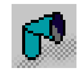
Defines the material of this JointElement (ADHA, ADHB or other).

Electrical Package

ElecBackShellE	ElecBppAttrE	ElecBundle
ElecBundleSegmentE	ElecCavity	ElecCommandSignal
ElecConShellE	ElecContactE	ElecCorrugateTubeE
ElecEqtPartE	ElecExtSpliceE	ElecFillerPlugE
ElecFnctCntPt	ElecFnctCnt	ElecFnctEqt
ElecGroundSignal	ElecGroupSignal	ElecIntSpliceE
ElecOffSheet	ElecPowerSignal	ElecShieldingSignal
ElecSicConE	ElecSignal	ElecSignalRoute
ElecStudE	ElecSystem	ElecTapeE
ElecTermBlockE	ElecTermination	ElecTerminationCst
ElecTermStripE	ElecVideoSignal	ElecWire

ElecBackShellE

Description

Describes the electrical feature of Back Shell type that you create when you click the  icon in the Electrical Library workbench.
For more information, refer to the *Electrical Library User's Guide*.

The back shell is a physical component used to guide the bundle segment extremity to the single insert connector, and to protect the crimping area.

Inheritance path: Standard - Feature -> ProductPackage - Product

Attributes

Elec_Extra_Length Type: **Double**

Defines the cable extra-length to be added to take into account the wire length inside the back shell.

Elec_Ref_Des Type: **String**

Defines the back shell reference designator attribute, which is the unique identifier for the back shell in the project.

Elec_Sub_Type Type: **String**

Defines the back shell subtype (User defined subtype).

ElecBppAttrE

Description

Describes the electrical feature of Bundle Segment Position Point type.
For more information, refer to the *Electrical Harness Installation User's Guide*.

The Bundle Segment Position Point type defines the point along a bundle segment at which the local slack is applied.

Inheritance path: Standard - Feature

Attributes

Elec_Slack Type: **Double**

Defines the slack length at the bundle segment position point.

ElecBundle

Description

Describes the electrical feature of Bundle type that you create when you click the  icon in the Electrical Wire Routing workbench.
For more information, refer to the *Electrical Wire Routing User's Guide*.

The ElecBundle type is an object that contains wires.

Inheritance path: Standard - Feature -> ProductPackage - Product

Attributes

Elec_Ref_Des Type: **String**

Defines the bundle reference designator attribute, which is the unique identifier for the bundle in the project.

Elec_Sub_Type Type: **String**

Defines the bundle subtype (User defined subtype).

ElecBundleSegmentE

Description

Describes the electrical feature of Bundle Segment type that you create when you click the  icon in the Electrical Harness Installation workbench.
For more information, refer to the *Electrical Harness Installation User's Guide*.

The ElecBundleSegmentE type is a segment of a geometrical bundle.

Inheritance path: Standard - Feature -> ProductPackage - Product

Attributes

Elec_FullConnected Type: **Boolean**
Is **True** if both bundle segment extremities are connected.

Elec_Bend_Radius Type: **Double**
Input data defining the bend radius value that corresponds to the minimum bend radius of the bundle segment curve.

Elec_Bend_Radius_OK Type: **Boolean**
Is **True** if the bundle segment real bend radius is greater than the **Elec_Bend_Radius** attribute.

Elec_Creation_Mode Type: **String**
Defines the electrical bundle segment creation mode. Three modes exist:

- **Slack:** Elec_Length is not valuated.
- **Bend:** Elec_Slack and Elec_Length are not valuated.
- **Length:** Elec_Slack is not valuated.

Elec_Di_Slack Type: **Double**
Input data defining the percentage of distributed slack along the bundle segment.
This attribute induces the value of the **Elec_Length_OUT** attribute.

Elec_Di_SlackOUT Type: **Double**
Output data valuated by the routing algorithm at the creation of the bundle segment in **Bend** or **Length** mode.
It defines the distributed slack.

Elec_Diameter Type: **Double**
Defines the bundle segment diameter.

Elec_Length Type: **Double**

Defines the bundle segment length: input data.

Elec_LengthOUT Type: **Double**

Output data valuated by the routing algorithm at the creation of the bundle segment in **Slack** or **Bend** mode. It defines the bundle segment length.

Elec_Segreg Type: **String**

Defines the bundle segment separation code used by the routing algorithm.

Elec_Sub_Type Type: **String**

Defines the bundle segment subtype.

ElecCavity

Description

Describes the electrical feature of Cavity type that you create when you click this icon  in the Electrical Library workbench.
For more information, refer to the *Electrical Library User's Guide*.

The cavity defines a reservation for a connector.

Inheritance path: Standard - Feature -> ProductPackage - Product

Attributes

Elec_Extra_Length Type: **Double**

Defines the wire length to be added to the wire routing length.

Elec_Id_Number Type: **String**

Defines a unique identifier for the cavity used to map a functional component and the corresponding physical part.

Elec_Number Type: **Double**

Defines the cavity number.

Elec_Ref_Des Type: **String**

Defines the cavity reference designator attribute, which is the unique identifier for the cavity in the project.

Elec_Sub_Type Type: **String**

Defines the cavity subtype.

ElecCommandSignal

Description

Describes the electrical feature of Command Signal type that you create when you click this icon in the Electrical System Functional Definition workbench.
For more information, refer to the *Electrical System Functional Definition User's Guide*.

The command signal is a logical connection between two or more components. It will be realized by a wire in physical world.

Inheritance path: Standard - Feature -> ProductPackage - Product -> Electrical - ElecSignal

Attributes

Elec_Nominal_Part_Num Type: **String**
Defines the nominal part number of the wire that realizes the command signal.

Elec_Recom_Wire_Type Type: **String**
Defines the attribute of the wire recommended to realize the signal.

Elec_Routing_Priority Type: **Double**
Defines the priority for the signal routing.

Elec_Sep_Code Type: **String**
Defines the separation code of the command signal used by the algorithm to find out the wire route.

Elec_Signal_Section Type: **Double**
Defines the command signal section.

Elec_Sub_Type Type: **String**
Defines the command signal subtype.

ElecConShellE

Description

Describes the electrical feature of Connector-Shell type that you create when you click the  icon in the Electrical Library workbench.
For more information, refer to the *Electrical Library User's Guide*.

A connector shell or shell is a non-electrical part which groups one or more electrical connector parts. It may be part of an equipment.

Inheritance path: Standard - Feature -> ProductPackage - Product

Attributes

Elec_Ref_Des Type: **String**

Defines the connector shell reference designator attribute, which is the unique identifier for the connector shell in the project.

Elec_Sub_Type Type: **String**

Defines the connector shell subtype.

ElecContactE

Description

Describes the electrical feature of Contact type that you create when you click the  icon in the Electrical Library workbench.
For more information, refer to the *Electrical Library User's Guide*.

A contact is an electrical component used within a termination and a cavity or between bundle segments.

Inheritance path: Standard - Feature -> ProductPackage - Product

Attributes

Elec_Barrel_Diameter Type: **Double**
Defines the hole diameter which lets the wire through.

Elec_External_Reference Type: **String**
Defines the contact reference from an external library.

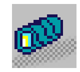
Elec_FullConnected Type: **Boolean**
Is **True** if all the contacts are connected.

Elec_Ref_Des Type: **String**
Defines the contact reference designator attribute, which is the unique identifier for the contact in the project.

Elec_Sub_Type Type: **String**
Defines the contact subtype.

ElecCorrugateTubeE

Description

Describes the electrical feature of Corrugated Tube type that you create when you click the  icon in the Electrical Library workbench. The corrugated tube is then instantiated using the Electrical Harness Installation workbench.
For more information, refer to the *Electrical Library* and *Electrical Harness Installation User's Guides*.

A corrugated tube is an electrical component applied onto bundle segments as a protection.

Inheritance path: Standard - Feature -> ProductPackage - Product

Attributes

Elec_Bend_Radius Type: **Double**

Defines the bend radius value, which corresponds to the minimum bend radius of the corrugated tube curve.

Elec_Bend_Radius_Protection_OK Type: **Boolean**

Is **True** if the **Elec_Bend_Radius** attribute is smaller than the real value of bend radius of the largest bundle segment linked to the corrugated tube.

Elec_Inner_Diameter Type: **Double**

Defines the corrugated tube inner diameter.

Elec_Length Type: **Double**

Defines the corrugated tube length.

Elec_Line_Type Type: **Double**

Defines the corrugated tube line type.

Elec_Line_Weight Type: **Double**

Defines the corrugated tube linear mass, used for the flattened representation.

Elec_Ref_Des Type: **String**

Defines the corrugated tube reference designator attribute, which is the unique identifier for the corrugated tube in the project.

Elec_Ref_PartNumber Type: **String**

Defines the corrugated tube reference part number.

Elec_Thickness Type: **Double**

Defines the corrugated tube thickness.

ElecEqtPartE

Description

Describes the electrical feature of Equipment type that you create when you click the  icon in the Electrical Library workbench.
For more information, refer to the *Electrical Library User's Guide*.

An equipment is an electrical device with one or more associated components: connectors, shells, contacts, filler plugs, placed in cavities. An equipment can also comprise terminations and bundle connection points.

Inheritance path: Standard - Feature -> ProductPackage - Product

Attributes

Elec_External_Reference Type: **String**

Defines the equipment reference from an external library.

Elec_Ref_Des Type: **String**

Defines the equipment reference designator attribute, which is the unique identifier for the equipment in the project.

Elec_Sub_Type Type: **String**

Defines the equipment subtype.

ElecExtSpliceE

Description

Describes the electrical feature of External Splice type that you create when you click the  icon in the Electrical Library workbench.
For more information, refer to the *Electrical Library User's Guide*.

An external splice is an electrical connector receiving bundle segments from different geometrical bundles.

Inheritance path: Standard - Feature -> ProductPackage - Product

Attributes

Elec_External_Reference Type: **String**
Defines the external splice reference from an external library.

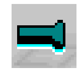
Elec_Ref_Des Type: **String**
Defines the external splice reference designator attribute, which is the unique identifier for the external splice in the project.

Elec_Sub_Type Type: **String**
Defines the external splice subtype.

Elec_FullConnected Type: **Boolean**
Is **True** if all the bundle connection points and terminations of the external splice are connected.

ElecFillerPlugE

Description

Describes the electrical feature of Filler Plug type that you create when you click the  icon in the Electrical Library workbench.
For more information, refer to the *Electrical Library User's Guide*.

A filler plug is an electrical component used to block up an unused cavity.

Inheritance path: Standard - Feature -> ProductPackage - Product

Attributes

Elec_Ref_Des Type: **String**

Defines the filler plug reference designator attribute, which is the unique identifier for the filler plug in the project.

Elec_Sub_Type Type: **String**

Defines the filler plug subtype.

ElecFnctCntPt

Description

Describes the electrical feature of Contact Point type that you create when you click the  icon in the Electrical System Functional Definition workbench.
For more information, refer to the *Electrical System Functional Definition User's Guide*.

This contact point is a functional electrical component that defines the point of contact or attachment for an electrical signal.

Inheritance path: Standard - Feature -> ProductPackage - Product

Attributes

Elec_Id_Number Type: **String**

Defines the contact point Id number.

Elec_Number Type: **Integer**

Defines a unique identifier for the contact point used to map a functional component and the corresponding physical part.

Elec_Ref_Des Type: **String**

Defines the contact point reference designator attribute, which is the unique identifier for the contact point in the project.

Elec_Signal_IO Type: **String**

Defines if the signal is input or output.

Elec_Signal_Unicity Type: **Boolean**

Defines the unicity of the signal: **True** if the signal is unique.

Elec_Sub_Type Type: **String**

Defines the contact point subtype.

ElecFnctCon

Description

Describes the electrical feature of Functional Connector type that you create when you click the icon in the Electrical System Functional Definition workbench.
For more information, refer to the *Electrical System Functional Definition User's Guide*.

This connector is a functional electrical component with one or more associated contact points, for example, a power or signal transmission connector.

Inheritance path: Standard - Feature -> ProductPackage - Product

Attributes

Elec_External_Reference Type: **String**

Defines the functional connector reference to an external library.

Elec_Id_Number Type: **String**

Defines a unique identifier for the functional connector used to map a functional component and the corresponding physical part.

Elec_Nominal_Part_Num Type: **String**

Defines the nominal part number of the physical connector that realizes the functional connector.

Elec_Ref_Des Type: **String**

Defines the functional connector reference designator attribute, which is the unique identifier for the connector in the project.

Elec_Sub_Type Type: **String**

Defines the functional connector subtype.

ElecFnctEqt

Description

Describes the electrical feature of Functional Equipment type that you create when you click the  icon in the Electrical System Functional Definition workbench.
For more information, refer to the *Electrical System Functional Definition User's Guide*.

This equipment is a functional electrical component with one or more associated connectors, for example a lamp or a battery.

Inheritance path: Standard - Feature -> ProductPackage - Product

Attributes

Elec_External_Reference Type: **String**

Defines the functional equipment reference to an external library.

Elec_Nominal_Part_Num Type: **String**

Defines the nominal part number of the physical equipment that realizes the functional equipment.

Elec_Ref_Des Type: **String**

Defines the functional equipment reference designator attribute, which is the unique identifier for the equipment in the project.

Elec_Sub_Type Type: **String**

Defines the functional equipment subtype.

ElecGroundSignal

Description

Describes the electrical feature of Ground Signal type that you create when you click this icon in the Electrical System Functional Definition workbench.
For more information, refer to the *Electrical System Functional Definition User's Guide*.

The ground signal is a logical connection between two or more components. It will be realized by a wire in physical world.

Inheritance path: Standard - Feature -> ProductPackage - Product -> Electrical - ElecSignal

Attributes

Elec_Ground_Unicity Type: **Boolean**
Defines the unicity of the ground signal: **True** if the signal is unique.

Elec_Nominal_Part_Num Type: **String**
Defines the nominal part number of the wire that realizes the ground signal.

Elec_Recom_Wire_Type Type: **String**
Defines the attribute of the wire recommended to realize the signal.

Elec_Routing_Priority Type: **Double**
Defines the priority for the signal routing.

Elec_Sep_Code Type: **String**
Defines the separation code of the ground signal used by the algorithm to find out the wire route.

Elec_Signal_Section Type: **Double**
Defines the ground signal section.

Elec_Sub_Type Type: **String**
Defines the ground signal subtype.

ElecGroupSignal

Description

Describes the electrical feature of Group Signal type that you create when you click this icon in the Electrical System Functional Definition workbench.
For more information, refer to the *Electrical System Functional Definition User's Guide*.

Groups signals will be routed together, for example shielded or twisted signals.

Inheritance path: Standard - Feature -> ProductPackage - Product -> Electrical - ElecSignal

Attributes

Elec_Nominal_Part_Num Type: **String**
Defines the nominal part number of the wire that realizes the group signal.

Elec_Recom_Wire_Type Type: **String**
Defines the attribute of the wire recommended to realize the signal.

Elec_Routing_Priority Type: **Double**
Defines the priority for the signal routing.

Elec_Sep_Code Type: **String**
Defines the separation code of the group signal used by the algorithm to find out the wire route.

Elec_Signal_Section Type: **Double**
Defines the group signal section.

Elec_Sub_Type Type: **String**
Defines the group signal subtype.

ElecIntSpliceE

Description

Describes the electrical feature of Internal Splice type that you create when you click the  icon in the Electrical Library workbench.
For more information, refer to the *Electrical Library User's Guide*.

An internal splice is a type of connector used to connect two or more wires belonging to the same bundle.

Inheritance path: Standard - Feature -> ProductPackage - Product

Attributes

Elec_External_Reference Type: **String**

Defines the internal splice reference to an external library.

Elec_Ref_Des Type: **String**

Defines the internal splice reference designator attribute, which is the unique identifier for the internal splice in the project.

Elec_Sub_Type Type: **String**

Defines the internal splice subtype.

ElecOffSheet

Description

Describes the electrical feature of Off Sheet Connector type that you create when you click the  icon in the Electrical System Functional Definition workbench.
For more information, refer to the *Electrical System Functional Definition User's Guide*.

An off sheet connector is a marker in the functional definition that is used to establish connections between different systems.

Inheritance path: Standard - Feature -> ProductPackage - Product

Attributes

Elec_Number Type: **Integer**

Defines the off sheet connector number.

Elec_Signal_IO Type: **String**

Defines if the signal is input or output.

Elec_Sub_Type Type: **String**

Defines the off sheet connector subtype.

ElecStudE

Description

Describes the electrical feature of Stud type that you create when you click this icon  in the Electrical Library workbench.
For more information, refer to the *Electrical Library User's Guide*.

A stud is an electrical connector receiving bundle segments with one or more wires connected through a termination. It is used to ground bundle segments or pieces of equipment.

Inheritance path: Standard - Feature -> ProductPackage - Product

Attributes

Elec_External_Reference Type: **String**
Defines the stud reference to an external library.

Elec_Ref_Des Type: **String**
Defines the stud reference designator attribute, which is the unique identifier for the stud in the project.

Elec_Sub_Type Type: **String**
Defines the stud subtype.

Elec_FullConnected Type: **Boolean**
Is **True** only if all the stud bundle connection points are connected.

ElecPowerSignal

Description

Describes the electrical feature of Power Signal type that you create when you click this icon in the Electrical System Functional Definition workbench.
For more information, refer to the *Electrical System Functional Definition User's Guide*.

The power signal is a logical connection between two or more components. It will be realized by a wire in physical world.

Inheritance path: Standard - Feature -> ProductPackage - Product -> Electrical - ElecSignal

Attributes

Elec_Nominal_Part_Num Type: **String**
Defines the nominal part number of the wire that realizes the power signal.

Elec_Nominal_Voltage Type: **Double**
Defines the power signal nominal voltage.

Elec_Recom_Wire_Type Type: **String**
Defines the attribute of the wire recommended to realize the signal.

Elec_Routing_Priority Type: **Double**
Defines the priority for the signal routing.

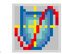
Elec_Sep_Code Type: **String**
Defines the separation code of the power signal used by the algorithm to find out the wire route.

Elec_Signal_Section Type: **Double**
Defines the power signal section.

Elec_Sub_Type Type: **String**
Defines the power signal subtype.

ElecShieldingSignal

Description

Describes the electrical feature of Shielding Signal type that you create when you click this icon  in the Electrical System Functional Definition workbench.
For more information, refer to the *Electrical System Functional Definition User's Guide*.

The shielding signal is a logical connection between two or more components. It will be realized by a wire in physical world.

Inheritance path: Standard - Feature -> ProductPackage - Product -> Electrical - ElecSignal

Attributes

Elec_Nominal_Part_Num Type: **String**
Defines the nominal part number of the wire that realizes the shielding signal.

Elec_Recom_Wire_Type Type: **String**
Defines the attribute of the wire recommended to realize the signal.

Elec_Routing_Priority Type: **Double**
Defines the priority for the signal routing.

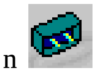
Elec_Sep_Code Type: **String**
Defines the separation code of the shielding signal used by the algorithm to find out the wire route.

Elec_Signal_Section Type: **Double**
Defines the shielding signal section.

Elec_Sub_Type Type: **String**
Defines the shielding signal subtype.

ElecSicConE

Description

Describes the electrical feature of Single Insert Connector type that you create when you click this icon  in the Electrical Library workbench.
For more information, refer to the *Electrical Library User's Guide*.

A single insert connector is an electrical connector male or female. It's the physical representation for both the plugs and the sockets.

Inheritance path: Standard - Feature -> ProductPackage - Product

Attributes

Elec_External_Reference Type: **String**

Defines the single insert connector reference to an external library.

Elec_Ref_Des Type: **String**

Defines the single insert connector reference designator attribute, which is the unique identifier for the single insert connector in the project.

Elec_Sub_Type Type: **String**

Defines the single insert connector subtype.

Elec_FullConnected Type: **Boolean**

Is **True** in only two cases:

- if the single insert connector is integrated into an equipment and connected to another single insert connector,
- if the single insert connector is connected to a bundle segment or a back shell and connected to another single insert connector.

ElecSignal

Description

Describes the electrical feature of Signal type that you create when you click the icon in the Electrical System Functional Definition workbench.
For more information, refer to the *Electrical System Functional Definition User's Guide*.

A signal is a logical connection between two or more components. May be of the following types: ground, shielding, video, power, command or grouped.

Inheritance path: Standard - Feature -> ProductPackage - Product

Attributes

Elec_Nominal_Part_Num Type: **String**
Defines the nominal part number of the physical wire that realizes the signal.

Elec_Recom_Wire_Type Type: **String**
Defines the attribute of the wire recommended to realize the signal.

Elec_Routing_Priority Type: **Double**
Defines the priority for the signal routing.

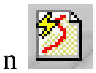
Elec_Sep_Code Type: **String**
Defines the separation code of the signal used by the algorithm to find out the wire route.

Elec_Signal_Section Type: **Double**
Defines the signal section.

Elec_ListPhysical Type: **CATIList**
Contains the list of **ElecWire** objects that realize the signal.

ElecSignalRoute

Description

Describes the electrical feature of Signal Route type that you create when you click this icon  in the Electrical Wire Routing workbench.
For more information, refer to the *Electrical Wire Routing User's Guide*.

The signal route is computed to find out the optimized way between two or more extremities of a signal.

Inheritance path: Standard - Feature

Attributes

Elec_Length Type: **Double**
Defines the signal route length.

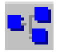
Elec_Nominal_Part_Num Type: **String**
Defines the nominal part number of the wire that realizes the signal.

Elec_Section Type: **Double**
Defines the signal route section.

Elec_Sub_Type Type: **String**
Defines the signal route subtype.

ElecSystem

Description

Describes the electrical feature of System type that you create when you click the  icon in the Electrical System Functional Definition workbench.
For more information, refer to the *Electrical System Functional Definition User's Guide*.

A system consists of equipments, connectors and signals. It is an electrical unit, which accomplishes a specific function.

Inheritance path: Standard - Feature -> ProductPackage - Product

Attributes

Elec_Ref_Des Type: **String**

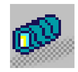
Defines the system reference designator attribute, which is the unique identifier for the system in the project.

Elec_Sub_Type Type: **String**

Defines the system subtype.

ElecTapeE

Description

Describes the electrical feature of Tape type that you create when you click the  icon in the Electrical Library workbench. The tape is then instantiated using the Electrical Harness Installation workbench.
For more information, refer to the *Electrical Library* and *Electrical Harness Installation User's Guides*.

A tape is an electrical component applied onto bundle segments as a protection.

Inheritance path: Standard - Feature -> ProductPackage - Product

Attributes

Elec_Bend_Radius_Delta Type: **Double**

Defines the bend radius value, which corresponds to the minimum bend radius of the tape curve.
This value takes into account the bundle segment and tape bend radius rule and ends up to an increased rigidity due to the tape.

Elec_Bend_Radius_Protection_OK Type: **Boolean**

Is **True** if the **Elec_Bend_Radius** attribute is smaller than the real value of bend radius of the largest bundle segment linked to the tape protection.

Elec_Covering_Length Type: **Double**

Defines the tape overlapping used when instantiating the protection.

Elec_Length Type: **Double**

Defines the tape length.

Elec_Line_Type Type: **Double**

Defines the tape line type, used for the flattened representation.

Elec_Line_Weight Type: **Double**

Defines the tape linear mass.

Elec_Number_Layer Type: **Double**

Defines the tape number of layers applied onto the bundle segment.

Elec_Ref_Des Type: **String**

Defines the tape reference designator attribute, which is the unique identifier for the tape in the project.

Elec_Ref_PartNumber Type: **String**

Defines the tape reference part number.

Elec_Tape_Thickness Type: **Double**

Defines the tape thickness.

Elec_Tape_Width Type: **Double**

Defines the tape width.

Elec_Taping_Angle Type: **Double**

Defines the taping angle.

Elec_Total_Tape_Length Type: **Double**

Defines the total tape length calculated according to the following formula:

$\sum$ **(NbLayers*NbWraps*3.1415*MaxDiameterValue)**

where:

NbWraps is the number of wraps of tape using as width, the tape width reduced by the overlapping value.
NbLayers is the number of layers
MaxDiameterValue is the bundle segment diameter for each segment.

Elec_Total_Thickness Type: **Double**

Defines the total tape thickness.

ElecTermBlockE

Description

Describes the electrical feature of Terminal Block type that you create when you click this icon  in the Electrical Library workbench.
For more information, refer to the *Electrical Library User's Guide*.

A terminal block is an electrical connector receiving bundle segments, each bundle segment being connected to a termination.

Inheritance path: Standard - Feature -> ProductPackage - Product

Attributes

Elec_External_Reference Type: **String**

Defines the terminal block reference to an external library.

Elec_Ref_Des Type: **String**

Defines the terminal block reference designator attribute, which is the unique identifier for the terminal block in the project.

Elec_Sub_Type Type: **String**

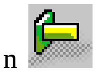
Defines the terminal block subtype.

Elec_FullConnected Type: **Boolean**

Is **True** if all the terminal block terminations are connected.

ElecTermination

Description

Describes the electrical feature of Termination type that you create when you click this icon  in the Electrical Library workbench.
For more information, refer to the *Electrical Library User's Guide*.

A termination is a sub-element ensuring the electrical signal conduction between any type of electrical component except the filler plug. It is indissociable from the electrical component and corresponds to a contact crimped into a cavity.

Inheritance path: Standard - Feature -> ProductPackage - Product

Attributes

Elec_Extra_Length Type: **Double**

Defines the length to be added to the wire routing length.

Elec_Id_Number Type: **String**

Defines a unique identifier for the termination used to map a functional component to the corresponding physical part.

Elec_Number Type: **Integer**

Defines the termination number.

Elec_Ref_Des Type: **String**

Defines the termination reference designator attribute, which is the unique identifier for the termination in the project.

Elec_Sub_Type Type: **String**

Defines the termination subtype.

ElecTerminationCst

Description

Describes the electrical feature of Termination type that you create when you click this icon  in the Electrical Library workbench.
For more information, refer to the *Electrical Library User's Guide*.

This type of termination only exists for terminal strip and the stud connectors. It has an associated geometry (a line), which allows the connection to be constrained between the bundle segment and the connector. The bundle segment can only be connected via this associated geometry (the line).

Inheritance path: Standard - Feature -> ProductPackage - Product

Attributes

Elec_Extra_Length Type: **Double**

Defines the length to be added to the wire routing length.

Elec_Id_Number Type: **String**

Defines a unique identifier for the termination used to map a functional component and the corresponding physical part.

Elec_Number Type: **Integer**

Defines the termination number.

Elec_Ref_Des Type: **String**

Defines the termination reference designator attribute, which is the unique identifier for the termination in the project.

Elec_Sub_Type Type: **String**

Defines the termination subtype.

ElecTermStripE

Description

Describes the electrical feature of Terminal Strip type that you create when you click this icon  in the Electrical Library workbench.
For more information, refer to the *Electrical Library User's Guide*.

A terminal strip is an electrical connector comprising a strip of terminations.

Inheritance path: Standard - Feature

Attributes

Elec_External_Reference Type: **String**
Defines the terminal strip reference to an external library.

Elec_Ref_Des Type: **String**
Defines the terminal strip reference designator attribute, which is the unique identifier for the terminal strip in the project.

Elec_Sub_Type Type: **String**
Defines the terminal strip subtype.

Elec_FullConnected Type: **Boolean**
Is **True** if all the terminal strip terminations are connected.

ElecVideoSignal

Description

Describes the electrical feature of Video Signal type that you create when you click this icon in the Electrical System Functional Definition workbench.
For more information, refer to the *Electrical System Functional Definition User's Guide*.

The video signal is a logical connection between two or more components. It will be realized by a wire in physical world.

Inheritance path: Standard - Feature -> ProductPackage - Product -> Electrical - ElecSignal

Attributes

Elec_Nominal_Part_Num Type: **String**
Defines the part number of the wire that realizes the video signal.

Elec_Recom_Wire_Type Type: **String**
Defines the attribute of the wire recommended to realize the signal.

Elec_Routing_Priority Type: **Double**
Defines the priority for the signal routing.

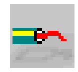
Elec_Sep_Code Type: **String**
Defines the separation code of the video signal used by the algorithm to find out the wire route.

Elec_Signal_Section Type: **Double**
Defines the video signal section.

Elec_Sub_Type Type: **String**
Defines the video signal subtype.

ElecWire

Description

Describes the electrical feature of Wire type that you create when you click the  icon in the Electrical Library workbench.
For more information, refer to the *Electrical Library User's Guide*.

Inheritance path: Standard - Feature -> ProductPackage - Product

Attributes

Elec_Bend_Radius Type: **Double**
Defines the bend radius.

Elec_CATALOG Type: **String**
Defines the catalog from which the wire is selected.

Elec_Color Type: **String**
Defines the color of the wire.

Elec_Diameter Type: **Double**
Defines the wire diameter.

Elec_FromConnectionPoint Type: **String**
Returns the reference designator value of the connection point to which the first extremity of the wire is connected.

Elec_FromDevice Type: **String**
Returns the reference designator value of the device to which the first extremity of the wire is connected.

Elec_FullConnected Type: **Boolean**
Is **True** if both wire extremities are connected.

Elec_IsNetworkConnex Type: **Boolean**
Is **True** if a route exists between whatever nodes only using the network connected branches.

Elec_IsRouted Type: **Boolean**
Is **True** if the wire is routed.

Elec_Length Type: **Double**

Defines the wire length.

Elec_Line_Weight Type: **Double**

Defines the wire linear mass.

Elec_Ref_Des Type: **String**

Defines the wire reference designator.

Elec_Sep_Code Type: **String**

Defines the separation code of the wire used by the algorithm to find out the wire route.

Elec_Signal Type: **ElecSignal**

Returns a product of type **ElecSignal** that realizes the wire.

Elec_Signal_Id Type: **String**

Defines the identifier of the signal used during the wire routing.

Elec_Sub_Type Type: **String**

Defines the wire subtype.

Elec_ToConnectionPoint Type: **String**

Returns the reference designator value of the connection point to which the second extremity of the wire is connected.

Elec_ToDevice Type: **String**

Returns the reference designator value of the device to which the second extremity of the wire is connected.

Elec_Cutting_Length Type: **Double**

Defines the wire routing length plus an extra length added for security when cutting the wire.

Elec_Shielding_Term Type: **String**

Defines a wire shielding type

FSAnalysis

FSCutPlaneAnalysis
FSInflectAnalysis
FSReflectAnalysis

FSCutPlaneAnalysis

Description

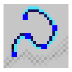
FSCutPlaneAnalysis describes a FreeStyle Analysis feature of Cutting Plane type which instance could be created selecting the  button.

Inheritance path

Standard ->Feature->BasicConstraintPackage->MfConstraint->FSAnalysis->FSCutPlaneAnalysis

FSInflectAnalysis

Description

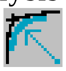
FSInflectAnalysis describes a FreeStyle Analysis feature of Inflection Line type which instance could be created selecting the  button.

Inheritance path

Standard->Feature->BasicConstraintPackage-MfConstraint ->FSSharedAnalysis ->FSAnalysis->FSInflectAnalysis

FSReflectAnalysis

Description

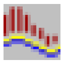
FSReflectAnalysis describes a FreeStyle Analysis feature of Reflection Line type which instance could be created selecting the  button

Inheritance path

Standard->Feature->BasicConstraintPackage->MfConstraint ->FSSharedAnalysis ->FSAnalysis ->FSReflectAnalysis

FSPrpnCurvature

Description

FSPrpnCurvature describes a FreeStyle Analysis feature of Porcupine Curvature type which instance could be created selecting the  button.

Inheritance path

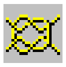
Standard->Feature->BasicConstraintPackage->MfConstraint->FSAnalysis->FSPrpnCurvature

FSConstraint

FSCntConstraint

FSCntConstraint

Description

FSCntConstraint describes a Shape feature of Continuity Constraint type which instance could be created selecting the  button

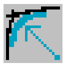
Inheritance path

Standard->Feature->FSCntConstraint

FSGeometry

FSCurveFillet

Description

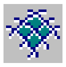
FSCurveFillet describes a curve Shape feature of Curve Fillet type which instance could be created selecting the  button.

Inheritance path

Standard->Feature->Visualizable->SkmDrwPackage->2DGeometry ->SkmDrwPackage->2DCurve ->FSCurveFillet

FSNet

Description

FSNet describes a surfacic Shape feature of FreeStyle Net type which instance could be created selecting the  button.

Inheritance path

Standard->Feature->Standard->Visualizable->MechanicalModeler ->GeometricFeature->MechanicalModelerHide->Body->BasicWireframePackage->Wireframe->BasicWireframePackage->Surface->FSNet

FSSweep

Description

FSSweep describes a surfacic Shape feature of FreeStyle Sweep type which instance could be created selecting the  button.

Inheritance path

Standard->Feature->Standard->Visualizable->MechanicalModeler ->GeometricFeature->MechanicalModelerHide->Body->BasicWireframePackage -> Wireframe->BasicWireframePackage->Surface->FSSweep

FSSharedAnalysis

FSAnalysis

Description

FSAnalysis is a global type for all FreeStyle Analysis features. No direct instance exists.

Inheritance path

Standard ->Feature->BasicConstraintPackage->MfConstraint ->FSAnalysis

FSCCKAnalysis

Description

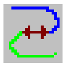
FSCCKAnalysis describes a FreeStyle Analysis feature of Connect Checker type which instance could be created selecting the  button.

Inheritance path

FSCCKAnalysis->FSSharedAnalysis ->FSAnalysis->BasicConstraintPackage->MfConstraint ->Standard->Feature

FSCrvCCKAnalysis

Description

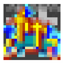
FSCrvCCKAnalysis describes a FreeStyle Analysis feature of Curve Connect Checker type which instance could be created selecting the  button.

Inheritance path

Standard->Feature->BasicConstraintPackage->MfConstraint->FSAnalysis ->FSCrvCCKAnalysis

FSDistAnalysis

Description

FSDistAnalysis describes a FreeStyle Analysis feature of Distance Analysis type which instance could be created selecting the  button.

Inheritance path

Standard->Feature->BasicConstraintPackage->MfConstraint->FSAnalysis->FSDistAnalysis

FSDraftAnalysis

Description

FSDraftAnalysis describes a FreeStyle Analysis feature of Draft Analysis type which instance could be created selecting the  button.

Inheritance path

Standard->Feature->BasicConstraintPackage->MfConstraint->FSAnalysis->FSDraftAnalysis

FSSurfCurvAnalysis

Description

FSSurfCurvAnalysis describes a FreeStyle Analysis feature of Surfacic Curvature type which instance could be created selecting the  button.

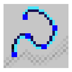
Inheritance path

Standard->Feature->BasicConstraintPackage->MfConstraint->FSAnalysis->FSSurfCurvAnalysis

FSSharedGeometry

FS3DCurve

Description

FS3DCurve describes a curve Shape feature of 3D Curve type which instance could be created selecting the  button.

Inheritance path

Standard->Feature->Standard->Visualizable->SkmDrwPackage->2DCurve->2DGeometry->FSSharedGeometry->FS3DCurve

FSUntrim

Description

FSUntrim describes a topological Shape feature of Untrim type, which instance could be created selecting the
 button.

Inheritance path

Standard->Feature->Standard->Visualizable->MechanicalModeler->GeometricFeature->MechanicalModelerHide->Body->BasicWireFramePackage->Wireframe->BasicWireFramePackage->Surface->FSUntrim

Functions Package

Analysis Operators	Generative Knowledge	KnowledgeExpertFunctions
Mathematical Functions	Measures	Measures Electrical
Messages and Macros	Part Measures	Space Analysis
Space Analysis (Interference checking)	String	Operators

Analysis Operators

- **bucklingfactors** (*Case: StaticSolution*)
  Computes a list of buckling factors.
  Example
  Bucklingfactors.1=BucklingFactors("Finite Element Model\Buckling Case Solution.1")

- **dispmax** (*Case: StaticSolution*)
  Computes the nodal maximum displacement.
  Example
  length.1=dispmax("Finite Element Model\Static Case Solution.1")

- **energy** (*Case: StaticSolution*)
  Computes the global energy in a static case solution.
  Example
  energy.1=energy("Finite Element Model\Static Case Solution.1")

- **frequencies** (*Case: StaticSolution*)
  Computes all the frequencies.
  Example
  FrequenciesList.1=Frequencies("Finite Element Model\Frequencies Case Solution.1")

- **frequency** (*Case: StaticSolution, Number: Integer*)
  Computes a given frequency.
  Example
  Frequency.1=Frequency("Finite Element Model\Frequency Case Solution.1")

- **globalerror** (*Case: StaticSolution*)
  Computes the global error percentage of a static case.
  Example
  percentage.1=globalerror("Finite Element Model\Static Case Solution.1")

- **misesmax** (*Case: StaticSolution*)
  Computes the maximum value of the nodal VonMises stress.
  Example
  misesmax.1=misesmax("Finite Element Model\Static Case Solution.1")

- **reaction** (*Entity: EntityForReaction, Case: StaticSolution, Axis:Axis System)*)
  Computes reactions on connections or boundary conditions.

GenerateScript()

Enables the user to launch a Generative Script from an Expert Rule.

Syntax

GenerateScript(E:\...\script.CATGScript","PartName","Function")

Where

- E:\...\script.CATGScript is the path of the .CATGScript
- PartName is the argument declared in the CATGScript file
- Function is the value of the argument

Example

```
if (pa.Name == pa.Name)
GenerateScript("..\..\script.CATGScript","PartName","Function","PartName2","MyPart")
```

Samples

Kwxscript.CATGScript and KwxPartGenerateScript.CATPart

To know more, see the *Generative Knowledge User's Guide*.

GetSubString()

Enables the user to extract a sub string from another string.

Syntax

GetSubString(String, index of the first character, number of characters to be extracted)

Mathematical Functions

- **abs**(Real): Real
  Calculates the absolute value of a number.

- **ceil**(Real): Real
  Returns the smallest integer value that is greater than or equal to the value specified in the argument.

- **floor**(Real):Real
  Returns the largest integer value that is less than or equal to the value specified in the argument.

- **int**(Real):Real
  Returns the integer value of a number.

- **let**
  Assigns a value to a temporary variable (let x = 30 mm)

- **min**(Real,Real):Real, **max**(Real,Real)
  Returns the minimum or maximum of a set of values specified in the argument.

- **sqrt**(Real):Real
  Returns the square root.

- **log**(Real):Real
  Returns the logarithm.

- **ln**(Real):Real
  Returns the natural logarithm.

- **round**(Real):Real
  Returns a rounded number.

- **exp**(Real):Real
  Returns the exponential.

- **LinearInterpolation**(*arg1*:Real, *arg2*:Real, *arg3*:Real) : Real
  Should be used when creating a parrallel curve from a law.
  <u>Example</u>:
  1 - Create a line in the Generative Shape Design workbench
  2 - Access the Knowledge Advisor workbench and create the law below:
  FormalReal.1 = LinearInterpolation(1,9,FormalReal.2)
  3 - Back to the Generative Shape Design, create a parralel curve. Select the Law mode and specify the law above as the one to be applied.

- **CubicInterpolation**(*arg1*:Real, *arg2*:Real, *arg3*:Real) : Real
  Should be used when creating a parrallel curve from a law.
  <u>Example</u>:
  1 - Create a line in the Generative Shape Design workbench
  2 - Access the Knowledge Advisor workbench and create the law below:
  FormalReal.1 = CubicInterpolation(1,50,FormalReal.2)
  3 - Back to the Generative Shape Design, create a parralel curve. Select the Law mode and specify the law above as the one to be applied.

- Mod() Enables the user to retrieve the rest of the division of the integer part of the real by the integer.

  Syntax: mod(Real,Integer):Real

  Sample: KwxMod.CATPart

Measures

Measures are functions that compute a result from data captured from the geometry area. Measures are application-related objects and they won't be displayed in the dictionary if you don't have the right product installed (Part Design or Generative Shape Design for example).

Sample: KwrMeasuresWiz.CATPart

- **distance(***Body1***,** *Body2***) :** Length
  Returns the distance between two bodies of a part.

Example:
Length.1 =
distance(Body.3 , Body.1)

- **length**(*GSMCurve*) :**Length**
  Returns the total length of a curve.

- **length**(*GSMCurve*, *Point1*, *Point2*) : **Length**
  Returns the length of a curve segment delimited by *Point1* and *Point2*.

- **length**(*GSMCurve*,*Point1*,*Boolean*): **Length**
  Returns the length of a curve segment located between *Point1* and one of the curve ends.
  Modifying the boolean value allows you to retrieve the length from the specified point to the other end.

- **area**(*Surface*): Area
  Returns the area of a surface generated by the Generative Shape Design product (an extruded surface for example).

- **area**(*Curve*) : Area
  Returns the area delimited by a curve.

- *point*.**coord**(*Integer*): Length
  Returns the coordinate of a point. Returns X if 1 is specified, Y if 2 is specified, Z if 3 is specified.

- *point*.**coord**(*oX*: Length, *oY*: Length, *oZ:* length): Void
  Assigns the point coordinates to the length parameters specified in the arguments. This method can only be used in Knowledge Advisor rules.

- Example:
  ```
  if Open_body.1\Point.2.coord(1) > 0mm
  Message("Point.2 abscissa is positive")
  else
  {
  Open_body.1\Point.1.coord(Xout, Yout, Zout)
  Message("Point.1 abscissa is: # ", Xout)
  }
  ```

- **volume**(*closedSurface*) : Volume
  Returns the volume of a closed surface.

- **angle**(*C*, *Point1*, *Point2*) : Angle
  Returns the angle between the lines "C-Point1" and "C-Point2".

- **angle**(*Line1*, *Line2*) : Angle
  Returns the angle between the *Line1* and *Line2* lines.

- **angle**(*direction1*,*direction2*) : Angle
  Returns the angle between two directions.

- *body*.**centerofgravity**(*oX*: Length, *oY*: Length, *oZ:* length): Void
  Assigns the values of the solid center of gravity coordinates to the parameters specified in the arguments. Cannot be used in a formula.

Example:
if Xout==2mm
Body.3.centerofgravity(Xcog,Ycog,Zcog)

- **curvature**(*crv*: Curve, *pt*: Point): Real
  Returns the curvature of a curve in a given point.

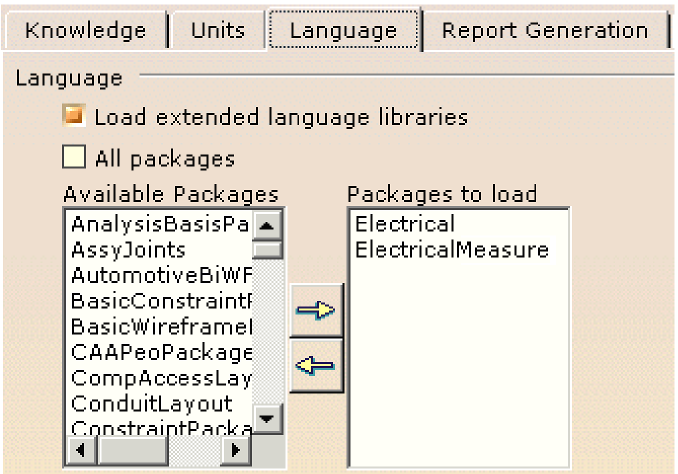
Example:
Real.1=
curvature(Open_body.1\Spline.1 ,Open_body.1\Point.2)

Electrical User Functions in Knowledge Products

About the Electrical User Functions...

To be able to use this function, you need to activate the **ElectricalMeasure** package.
To do so:

- select **Tools -> Options... -> General -> Parameters and Measures** and go to the **Language** tab.
- choose **ElectricalMeasure** and click the right arrow:



- click **OK** to validate.

ElecDistanceCommon

Syntax

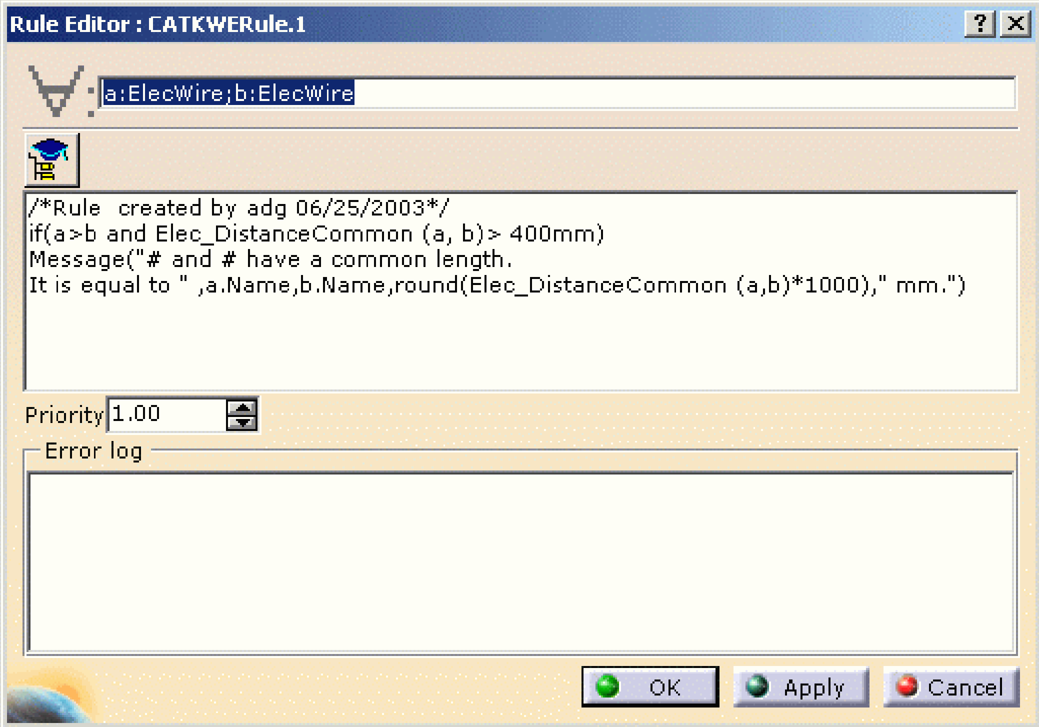
ElecDistanceCommon(*Wire1:Feature,Wire2:Feature*):**Length**

Returns the common length of the two wires given as input arguments.

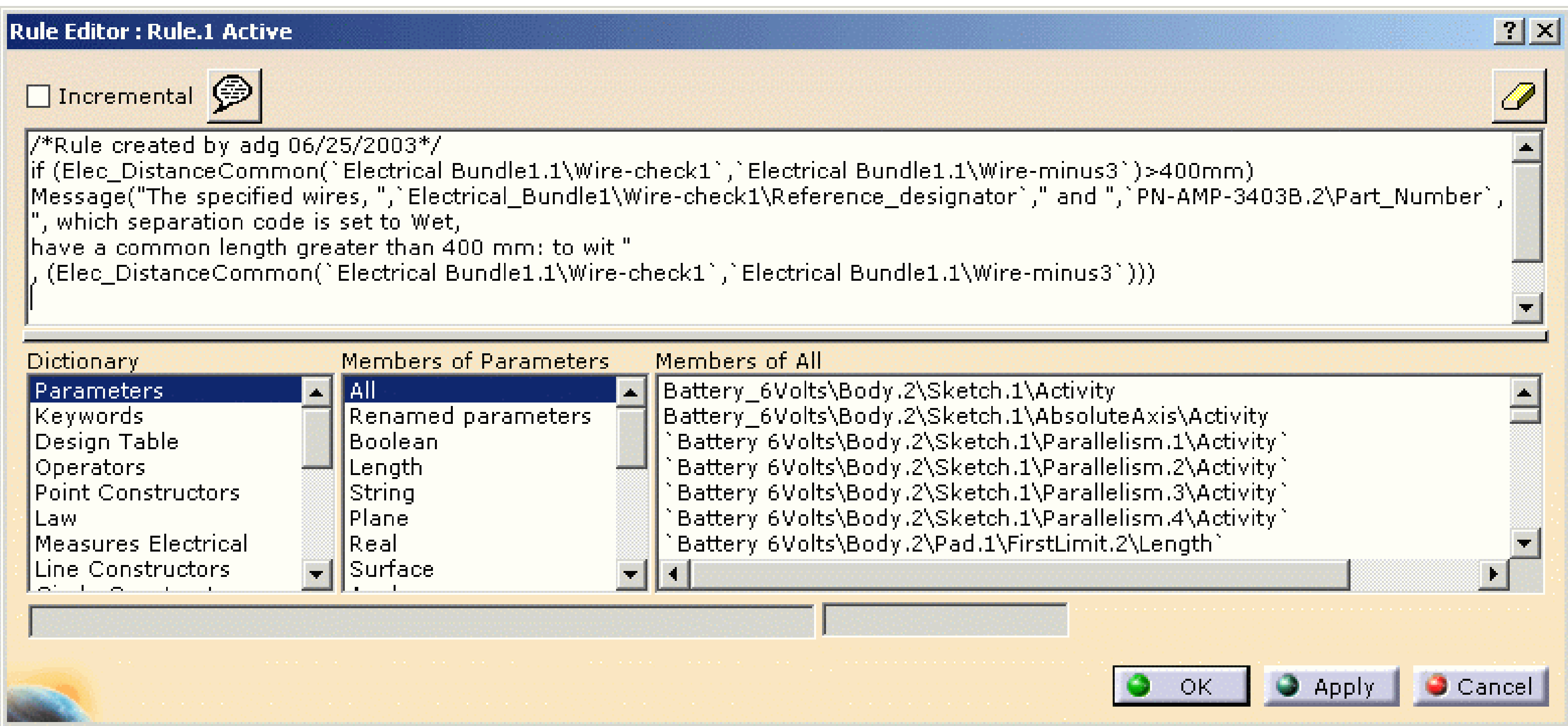
The type of Wire1 and Wire2 is ElecWire.

Examples

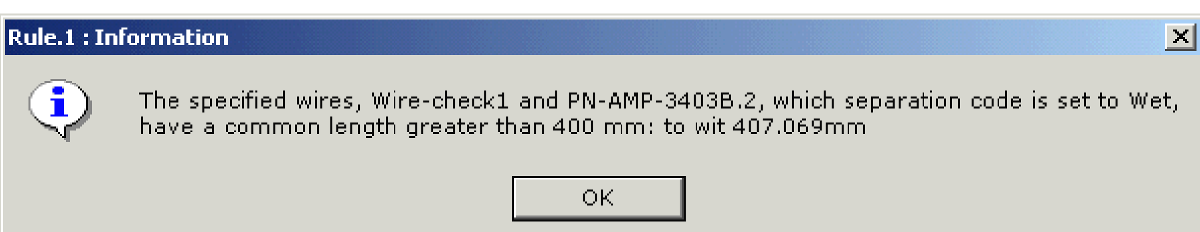
The **ElecDistanceCommon** user function can be used in Knowledge Expert to find all the couples of wires in the session that have a common length greater than a given value.



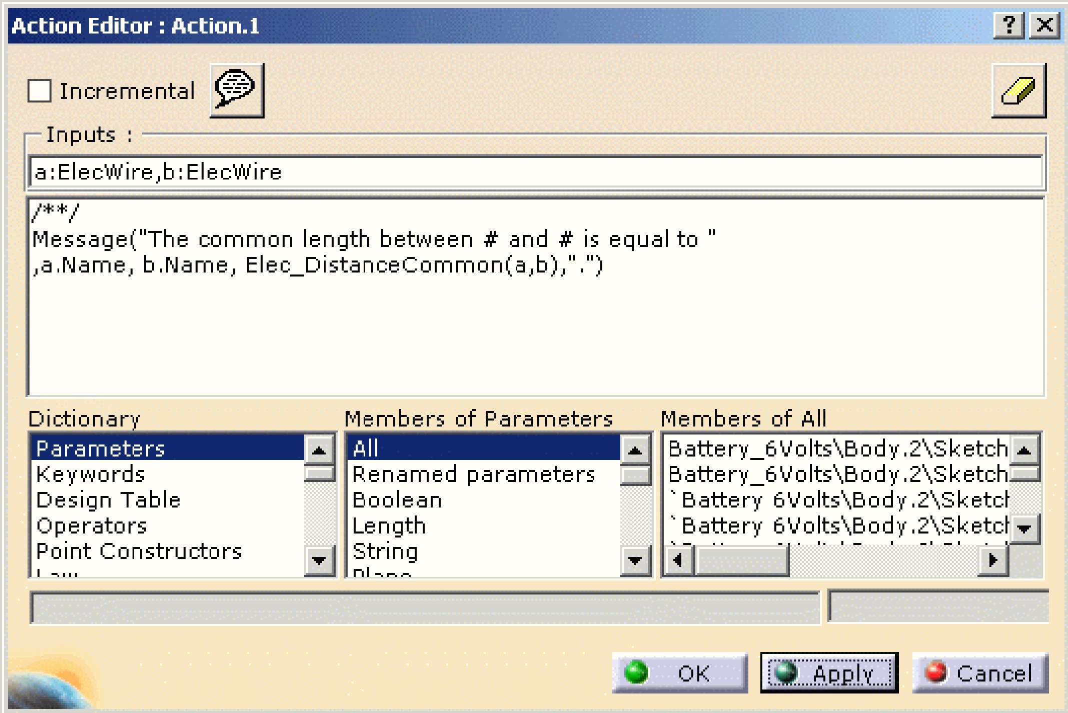
In Knowledge Advisor, it can be used to define a rule giving the common length of two specific wires sharing properties.



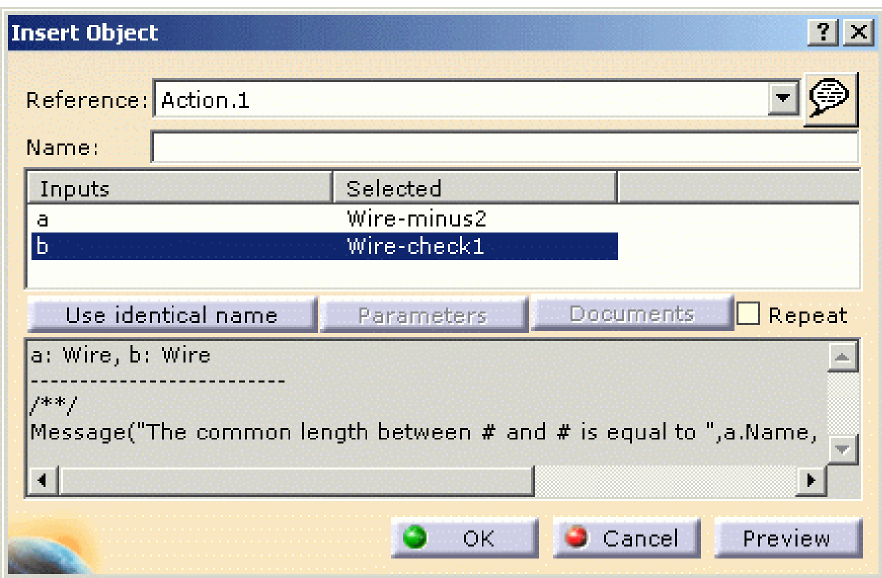
Applying the rule displays the following message if the condition is met:



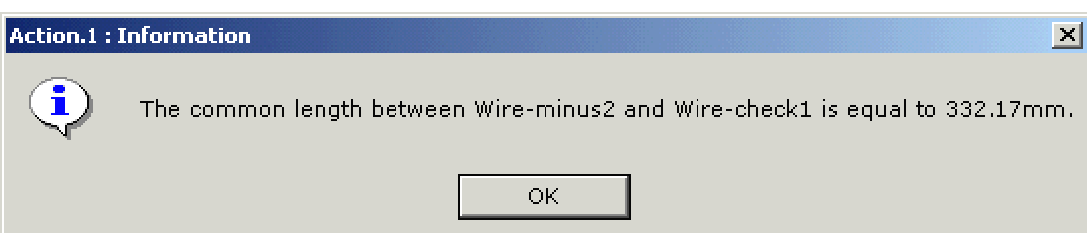
Still in Knowledge Advisor, to verify that two wires selected in the specification tree have a common length, the following action can be defined:



then ran: select two wires in the specification tree and click OK to validate.



The following message displays:



Messages and Macros

Message Function	Question Function
LaunchMacroFromDoc Function	LaunchMacroFromFile Function

Message Function

Displays a message in an information box. The message can include one or more parameter values.

Syntax

Message(*String [# String1 # String2 ..., Param1Name, Param2Name, ...]* **) :** Void

The **Message** function takes one required argument and several optional arguments depending on whether parameter values are to be displayed in the message.

Arguments	**Description**
String	Required. String to be displayed in the information box (should be put in quotes).
# String1, Param1Name...	Optional. When parameter values are to be displayed within the message, the arguments should be specified as follows: • one string in quotes including a # symbol wherever a parameter value is to be displayed • as many [, parameter name] statements as parameter values declared with a "#" in the message.

Use the "\|" symbol to insert a carriage return in a message.

Example

```
Message("External radius is: # | Internal Radius is: #",
PartBody\Sketch.1\Radius.3\Radius,
PartBody\Hole.1\Diameter)
```

LaunchMacroFromDoc Function

Executes a macro stored in a document from a rule.
A macro is stored in a document when you don't specify any external file before recording it.

Warning: It is up to the user to check that the macro which is run is not going to cause an infinite loop or result in a system crash.

Syntax

LaunchMacroFromDoc(*MacroName* **)**

The Macro Name should be put between quotes

Example

```
LaunchMacroFromDoc("Macro1")
```

LaunchMacrofromFile Function

Executes a macro CATScript from a rule.

Warning: It is up to the user to check that the macro which is run is not going to cause an infinite loop or result in a system crash.

Syntax

LaunchMacroFromFile("*MacroName*.CATScript" **)**

Example

```
LaunchMacroFromFile("Macro1.CATScript")
```

Question Function

P2

Displays a message in a dialog box, waits for the user to click a button and returns a value indicating which button the user clicked (true if Yes was clicked, false if No was clicked)

Syntax

Question(*String [# String1 # String2 ..., Param1Name, Param2Name, ...]* **):** Boolean

The **Question** function takes one required argument and several optional arguments depending on whether parameter values are to be displayed in the message.

Arguments	***Description***
String	Required. String to be displayed in the dialog box (should be put in quotes).
# String1, Param1Name...	Optional. When parameter values are to be displayed within the message, the arguments should be specified as follows: • one string in quotes including a # symbol wherever a parameter value is to be displayed • as many [, parameter name] statements as parameter values declared with a "#" in the message.

Use the "|" symbol to insert a carriage return in a prompt.

Example

```
Boolean2 =
Question("SketchRadius is # | Do you want to change this value ?",
PartBody\Sketch.1\Radius.3\Radius )
```

Operators

Arithmetic operators

+ Addition operator (also concatenates strings)

- Subtraction operator

* Multiplication operator

/ Division operator

= Assignment operator

** Exponentiation operator

Comparison Operators

<> Not equal to

== Equal to

>= Greater or equal to

<= Less than or equal to

< Less than

> Greater than

Part Measures

- **area()**
  Returns the area of an object of CATFace type.

  Syntax: **area**(CATFace) : Area

- **length()**
  Returns the length of an object of CATEdge type (the edge of a cube, or the length of a spline for example).

  Syntax: **length**(CATEdge) : Length

- **smartVolume**
  Returns the volume of a solid.

  Syntax: smartVolume(elem: Solid, ...): Volume

- **smartWetarea()**
  Returns the wet area of a solid.

  Syntax: smartWetarea(elem: Solid, ...): Area

smartVolume and smartWetarea refer to intermediate states of a solid. smartVolume does not compute the volume of each pad contained in a PartBody but the total volume.
Example: Given a PartBody containing 3 pads: The volume of Pad.1 = 0.1m3, The volume of Pad.2=0.1m3 and the volume of Pad.3=0.1m3. The Volume of Pad.3 displayed will be Pad.3=0.3M3: The volume of Pad.3=the Volume of Pad.1+ the volume of Pad.2.

Note that this applies also to smartWetarea (the total wet area is computed).

Space Analysis

ClashOrContact	DistanceMin
DistanceMinXYZ	IsIncludedIn
PenetrationMax	

ClashOrContact

Note about the Clash functions

Syntax

ClashOrContact(*String*,*Product1*,*Product2*):**Boolean**
Determines whether two components are clashing or contacting. The first argument is either "Clash" or "Contact" depending on the type of analysis you want to be performed.

Example

Activate the ClashRule1and ClashRule2 rules in KwxClash.CATProduct and run the Force Solve command from the Rule Base contextual menu.

The clashing components are:
p1 with p2
p2 with p3
p5 with p2
p5 with p1.

The components in contact are:
p3 and p2
p5 and p2.

DistanceMin

Syntax

DistanceMin(*Product1*,*Product2***):Length**
Returns the minimum distance between two components.

Example

Activate the DistanceMinCheck check in KwxClash.CATProduct and run the Force Solve command from the Rule Base contextual menu.
The components which are distant from one another of more than 0.02mm are:
p3 and p1
p4 and p3
p4 and p2
p5 and p4
p5 and p3.

DistanceMinXYZ

Note about the Space Analysis functions

Syntax

DistanceMin(String,***Product1*,*Product2*****):Length**
Returns the minimum distance along a direction between two components. The first argument is either "X", "Y" or "Z" depending on the direction.

IsIncludedIn

Note about the Clash functions

Syntax

IsIncludedIn(*Product1*,*Product2***):Boolean**
Determines whether a component is included in another.

Example

Activate the ClashCheck1 check in KwxClash.CATProduct and run the Force Solve command from the Rule Base contextual menu.
p1 is included in p2
p5 is included in p1.

PenetrationMax

Note about the Clash functions

Syntax

PenetrationMax(*Product1*,*Product2***):Length**
Returns the maximum length of one component within another.

Example

Activate the PenetrationMaxCheck check in KwxClash.CATProduct and run the Force Solve command from the Rule Base contextual menu.
p2 penetration into p1 is of 0.03mm.

DefineInterferenceComputation

This function is available if the SpaITFCheckMethod package is loaded (**Tools -> Options... -> Parameters and Measure -> Language tab**). The interference computation is run from the Space Analysis workbench.

Syntax

DefineInterferenceComputation(*p1:Product, p2:Product, TypCalc: String, ClearVal: Length, NameShape1: String, NameShape2: String, ThisRule: KWERule*)

Defines the interference type, clearance value and shapes to be used in the interference computation between a pair of products.

where *TypCalc* is the interference type and *ClearVal* the clearance value in MKS units.

Example

```
if (p1 != p2)
{
 DefineInterferenceComputation (p1, p2,
                                "Clearance", 70mm,
                                "WRAPPING", "Shape 1",
                                 ThisRule);
}
```

String

BuildMessageNLS

Enables the user to send messages or ask qusetions throught the Message and Question functions in the language of his choice. The BuildMessageNLS function can build a NLS message (a message in his own language) by finding it in a CATXXX.CATNls file.

Note that this function is useful when used together with the Message and Question functions. To know more about these 2 functions, see the Knowledge Advisor documentation.

Syntax: BuildMessageNLS(*MessageCatalog:String*, *MessageKey: String*, *argument: Literal*, ...):String

where

MessageCatalog:*String* is the name of the CATXXX.CATNls file where we will find the NLS message (in fact, it is the CATXXX name without the CATNls extension).

MessageKey: *String* is the key name in this catalog

argument: *Literal*, ... are values that will be replaced in the message.

Example:

The KwrCATCatalog.CATNls file contains the following text.

```
Zero = "Zero";

Un = "Un /P1";

Deux = "Deux /P1 /P2";
```

Zero Un and Deux are the messages. The first message has no arguments, the second has 1 argument, the third, 2 arguments.

To display those messages in a Knowledge Advisor rule for example, write the following rule body:

Message (BuildMessageNLS("KwrCATCatalog","Zero"))

Where x,y,z are parameters.

OR

Message (BuildMessageNLS("KwrCATCatalog","Un",x))

Where x,y,z are parameters.

OR

Message (BuildMessageNLS("KwrCATCatalog","Deux",y,z))

Where x,y,z are parameters.

- Note that if the function does not find the key or the .NLS catalog, it will return an empty string.
- If there are too many parameters compared to the number of arguments of the message, the parameters will be ignored.
- If there are too few parameters compared to the number of arguments of the message, the parameters will be replaced by a "???" string.
- Note that the .NLS file is to be stored in the runtime view (in the msgcatalog directory)

Sample: KwrCATCatalog.CATPart (See Rule.2)

ReplaceSubText Function

Replaces a substring with another substring within a character string

Syntax

ReplaceSubText(*InputString:* String, *SubStringToBeReplaced:* String, *ReplacingSubString:* String**):** String

Arguments 2 and 3 can be specified either with their parameter names or with the string itself between quotes.

ToUpper Function

Changes all lower-case letters of a string to upper-case.

Syntax

ToUpper(*StringTobeConverted***:** *String***):** String

where *StringTobeConverted* is name of the string type parameter.

ToLower Function

Changes all upper-case letters of a string to lower-case.

Syntax

ToLower(*StringTobeConverted***:** *String***):** String

where *StringTobeConverted* is name of the string type parameter.

ToString Function

Converts an integer into a string.

Syntax

ToString(*Integer***) : String**

Knowledge Advisor

AdvisorAction	AdvisorCheck
AdvisorConnection	AdvisorFeature
AdvisorFormula	AdvisorLaw
AdvisorMacrosSet	AdvisorParameterSet
AdvisorReaction	AdvisorRelation
AdvisorRelationSet	AdvisorRootRelation
AdvisorRule	AdvisorSetofEquations
DesignTableType	DocumentTemplate
DTLotusSheetType	DTModelSheetType
DTSheetType	DTTextSheetType
KWANamedURL	VBScript
Loop	

AdvisorAction

Description

Type describing the actions created by the user when clicking the Actions icon () in Knowledge Advisor.

Inheritance path

Standard->Feature->KnowledgeAdvisor->AdvisorFeature->AdvisorConnection->AdvisorAction

Attributes

Body

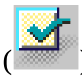
Describes the body of the action. Note that this attribute is an internal representation.

Signature

Describes the declaration of formal parameters.

AdvisorCheck

Description

Type describing the checks created by the user when clicking the Check icon () in the Knowledge Advisor workbench.

Inheritance path

Standard->Feature->KnowledgeAdvisor->AdvisorFeature->AdvisorConnection->AdvisorRootRelation->AdvisorRelation->AdvisorCheck

Attributes

Result
Describes the result of the check in the form of parameters (that can be used in a formula).

Severity
Describes the type of the check (silent, warning, or information.)

Warning
Describes the message that will display when the check is launched.

AdvisorConnection

Description

Type describing an object that references other objects. This type cannot be instantiated.

Inheritance path

Standard->Feature->KnowledgeAdvisor->AdvisorFeature->AdvisorConnection

Attributes

Parameters

Describes the list of parameters used by the relation.

AdvisorFeature

Description

Type describing all Knowledge Advisor features:

- Parameters
- Formulas
- Relations ...

Inheritance path

Standard->Feature->AdvisorFeature

Attributes

Constant
Indicates if the relation is considered as constant or not.

Hidden
Indicates if the relation is hidden or not.

AdvisorFormula

Description

Type describing the formulas created by the user when clicking the Formula icon ().

Inheritance path

Standard->Feature->KnowledgeAdvisor->AdvisorFeature->AdvisorRootRelation
->AdvisorRelation->AdvisorFormula

AdvisorLaw

Description

Type describing the law created by the user when clicking the Law icon in Knowledge Advisor.

Inheritance path

Standard->Feature->KnowledgeAdvisor->AdvisorFeature->AdvisorConnection->AdvisorLaw

Attributes

Body
Corresponds to the body of the law. Note that this attribute is an internal representation.

Formal Parameters
Corresponds to the formal parameters field of the Law dialog box.

AdvisorMacrosSet

Description

Type describing the macros with arguments created by the user when clicking the Macro with arguments icon () in the Knowledge Advisor workbench.

Inheritance path

Standard->Feature->KnowledgeAdvisor->AdvisorFeature->AdvisorMacrosSet

AdvisorParameterSet

Description

Type describing the sets of parameters created by clicking the Add Set of Parameters icon ( ) in the Knowledge Advisor workbench.

Inheritance path

Standard->Feature->KnowledgeAdvisor->AdvisorFeature->AdvisorParameterSet

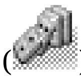
Attributes

Parameters
Enables the user to add parameters.

ParameterSets
Enables the user to sets of parameters.

AdvisorReaction

Description

Type describing the reactions created by the user when clicking the Reaction icon () in the Knowledge Advisor workbench.

Inheritance path

Standard->Feature->KnowledgeAdvisor->AdvisorFeature->AdvisorRootRelation->AdvisorReaction

Attributes

EventsToReact
Describes the events available in the Knowledge Advisor workbench.

ReactionAction
Enables the user to specify if he wants to write the action in VB or in the knowledgeware language.

ReactionType
Enables the user to choose what type of reaction will be fired when one of the events occurs.

SourceList
Enables the user to select a source in the geometry or in the specification tree.

TypeSource
Corresponds to the Source Type in the Reaction dialog box. This field enables the user to select Selection or Owner. In the first case, he can manually select one or more items in the specification tree or in the geometrical area. In the second case, he links the action with a feature of the geometry or of the specification tree

AdvisorRelation

Description

Type describing all relations.

Inheritance path

Standard->Feature->KnowledgeAdvisor->AdvisorFeature->AdvisorRootRelation->AdvisorRelation

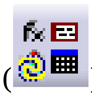
Attributes

Body

Describes the body of the relation. Note that this attribute is an internal representation.

AdvisorRelationSet

Description

Type describing the sets of relations created when clicking the Add Set of Relations icon () in the Knowledge Advisor Workbench.

Inheritance path

Standard->Feature->KnowledgeAdvisor->AdvisorFeature->AdvisorRelationSet

AdvisorRootRelation

Description

Type describing all Knowledge Advisor relations.

Inheritance path

Standard->Feature->KnowledgeAdvisor->AdvisorConnection->AdvisorRootRelation

Attributes

Activity

Indicates if the relation is enabled (true) or not (false).

AdvisorRule

Description

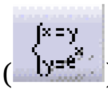
Type describing the rules created by the user when clicking the Advisor Rule icon () in the Knowledge Advisor workbench.

Inheritance path

Standard->Feature->KnowledgeAdvisor->AdvisorFeature->AdvisorRootRelation->AdvisorRelation->AdvisorRule

AdvisorSetOfEquations

Description

Type describing the sets of equations created by the user when clicking the Set of Equations icon () in Knowledge Advisor.

Inheritance path

Standard->Feature->KnowledgeAdvisor->AdvisorFeature->AdvisorRootRelation ->AdvisorRelation->AdvisorSetOfEquations

Attributes

BlackBoxAttempts

This option is only used for sets of constraints with more than one variable. This option changes the number of start points used by the solver. For instance, for a 2-dimensional problem, the following number of calls will be performed:
Number of attempts: 1-->3 start points
Number of attempts: 2-->9 start points
Number of attempts: 3-->27 start points

BlackBoxCacheSize

Enables the user to choose the size of internal black boxes caches. Decrease this value only if you do not have enough memory. Increase this value if you have enough memory in order to reduce the computation time

BlackBoxMaxCallsNB

Enables the user to limit the number of measures calculation (equivalent to a time limit.)

BlackBoxUnimodal

Enables the user to use special information about the interval of unimodality.

IsStopDialogDisplayed

Enables the user to display a "Stop" dialog box that will enable the user to interrupt the computation.

MaxCalculationTime

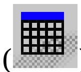
Enables the user to indicate the computation time. If the indicated time is equal to 0, the computation will last until a solution is found.

Precision

Enables the user to define the precision of the results.

DesignTableType

Description

Type describing the design tables created by the user when clicking the Design Table icon () in the Knowledge Advisor workbench.

Inheritance path

Standard->Feature->KnowledgeAdvisor->AdvisorFeature->AdvisorRootRelation
->DesignTableType

Attributes

Associations
Describes the association of the names of the columns and the parameters driven by the design table.

ConfigurationRow
Indicates the line number of the design table used to valuate parameters.

HiddenColumns
Describes the hidden columns of the design table.

Sheet
Indicates the sheet number of the design table (Excel only).

DocumentTemplate

Description

Type describing the document templates created by the user in the PKT workbench.

Inheritance path

Standard->Feature->KnowledgeAdvisor->AdvisorFeature->DocumentTemplate

DTLotusSheetType

Description

Type describing the Lotus design tables.

Inheritance path

Standard->Feature->KnowledgeAdvisor->AdvisorFeature->DTSheetType->DTLotusSheetType

DTModelSheetType

Description

Type describing the sheet stored in the model. In this case, the results are stored in the model.

Inheritance path

Standard->Feature->KnowledgeAdvisor->AdvisorFeature->DTSheetType->DTModelSheetType

DTSheetType

Description

Type describing the Excel design tables.

Inheritance path

Standard->Feature->KnowledgeAdvisor->AdvisorFeature->DTSheetType

Attributes

ColumnsNb
Indicates the number of columns of the design table.

CopyMode
If set to 1, the data are stored in the model and in the file.
If set to 0, the data are stored in the file only.

RowNb
Indicates the number of rows.

SheetCopy
When CopyMode is set to 1, describes the file content.

SheetIndex
Indicates the sheet number in Excel.

VerticalColumns
Indicates if the columns are vertical or horizontal in Excel.

DTTextSheetType

Description

Type describing the .txt design tables.

Inheritance path

Standard->Feature->KnowledgeAdvisor->AdvisorFeature->DTSheetType->DTTextSheetType

KWANamedURL

P2

Description

Type describing the URL that the user can add to a relation by clicking the Comment and URLs icon in the Knowledge Advisor workbench.

Inheritance path

Standard->Feature->KWANamedURL

Attributes

URLLocation
Indicates the URL.

URLName
Indicates the name associated to the URL.

Loop

Description

Describes the loop that can be created in Knowledge Advisor.

Attributes

Action: Script enabling the user to define what he creates.
Example: "mypad isa pad"

ActionsContext: Enables the user to define the creation context.

Arguments: Not available.

FirstItemForAction: First value of the "i" iterator.

InIteratorContext: Not available.

Iterator: Current iterator.

LastItemForAction: Last value of the "i" iterator.

Values: Not available

To know more see, the Knowledge Advisor documentation.

VBScript

Description

Type describing the scripts created by the user when clicking the VBScript icon.

Inheritance path

Standard->Feature->KnowledgeAdvisor->AdvisorFeature->VBScript

Attributes

Arguments

Enables the user to enter arguments.

ScriptText

Enables the user to enter the script.

Knowledge Expert

KWERuleBaseComponent	KWERuleBase
KWECheck	KWERule
KWERuleSet	KWEGenericRuleBaseComponent

KWERuleBaseComponent

Description

Type enabling the user to access the *For all* field or the Body of the rule or the check in Read only mode.

Inheritance path

Standard->Feature->KnowledgeExpert->KweGenericRuleBaseComponent

Attribute

BodyString:
Contains the check or rule body.

Language:
Corresponds to the language of the check: 1 for the Knowledge Expert language and 2 for Visual Basic.

VariableString:
Variable specified in the *For all* field.

KWERuleBase

Description

Type describing the rule base created when accessing the Knowledge Expert icon.

Inheritance path

Standard->Feature->KnowledgeExpert->KweGenericRuleBaseComponent->KnowledgeExpert->KweRuleSet

KWECheck

Description

Type describing the checks created by the user when clicking the Expert Check icon.

Inheritance path

Standard->Feature->KnowledgeExpert->KWEGenericRuleBaseComponent->KnowledgeExpert->KWERuleBaseComponent

Attributes

CorrectFunction
Enables the user to access the Correct tab of the Check editor.

CorrectFunctionComment
Enables the user to access the Correction Comment section of the Check editor.

CorrectFunctionType
Real enabling the user to access the various types of correct functions.

Help
Enables the user to access the Help section of the Correct tab of the Check editor.

KWERule

Description

Type describing the rules that the user creates when clicking the Expert Rule icon.

Inheritance path

Standard->Feature->KnowledgeExpert->KWEGenericRuleBaseComponent->KnowledgeExpert->KWERuleBaseComponent

Attribute

Priority:
Indicates the rule priority level.

KWERuleSet

Description

Type describing the rule set that the user creates when clicking the Rule set icon or when creating an Expert Check or Rule.

Inheritance path

Standard->Feature->KnowledgeExpert->KWEGenericRuleBaseComponent

Attribute

DirectChildren
Attribute of List type enabling the user to retrieve the items located below the ruleset.

KWEGenericRuleBaseComponent

P2

Description

Type enabling the user to reference the Knowledge Expert objects.

Inheritance path

Standard->Feature

Attribute

Activated:

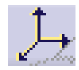
Indicates if the sub-components of the RuleBase are activated.

Comment:

Shows the comments appearing when creating the expert rule or the expert check and that can be modified using the Edit->Properties menu in the Knowledge Expert workbench.

MechanicalModeler Package

AxisSystem

Describes the feature that you create when you select the  icon in the Part Design workbench.

- IsDatum: If set to true, corresponds to a datum.
- IsLeaf: If set to true, corresponds to a geometrical object without any father object.
- IsRoot: If set to true, corresponds to a geometrical object without any son object.

BodyFeature

Describes the feature that you create when you select the **Insert ->Body** command from the menu in the Part Design workbench.

GeometricFeature

Describes a geometric feature: point, plane,...

- IsDatum: If set to true, corresponds to a datum.
- IsLeaf: If set to true, corresponds to a geometrical object without any father object.
- IsRoot: If set to true, corresponds to a geometrical object without any son object.

Mechanical Feature

Describes a feature created in the Part Design workbench (pad, ...).

- IsDatum: If set to true, corresponds to a datum.
- IsLeaf: If set to true, corresponds to a geometrical object without any father object.
- IsRoot: If set to true, corresponds to a geometrical object without any son object.

OpenBodyFeature

Describes the feature that you create when you select the **Insert ->OpenBody** command in the Generative Shape Design workbench.

PartFeature

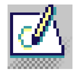
Describes a part.

- IsDatum: If set to true, corresponds to a datum.
- IsLeaf: If set to true, corresponds to a geometrical object without any father object.
- IsRoot: If set to true, corresponds to a geometrical object without any son object.

PowerCopy

Describes the feature that you create when you select the **Insert ->Advanced Replication Tools -> PowerCopy Creation...** command from the menu.

SketchFeature

Describes the features that you create when clicking the  icon to access the Sketcher in the Part Design workbench.

SkinFeature

Describes a skin feature.

- IsDatum: If set to true, corresponds to a datum.
- IsLeaf: If set to true, corresponds to a geometrical object without any father object.
- IsRoot: If set to true, corresponds to a geometrical object without any son object.

UserFeature

Describes the feature that you create when you select the **Insert ->UserFeature -> UserFeature Creation...** command from the menu.

- IsDatum: If set to true, corresponds to a datum.
- IsLeaf: If set to true, corresponds to a geometrical object without any father object.
- IsRoot: If set to true, corresponds to a geometrical object without any son object.

For more information, refer to the *Product Knowledge Template User's Guide*.

Optimization

FullDOEAlgorithm	OptApproximationGradientAlgorithm
OptConstraint	OptConstraintSatisfaction
OptFeature	OptFreeParameter
OptGenericAlgorithm	OptGenericDOEAlgorithm
OptGenericOptimAlgorithm	OptGoal
OptGradientAlgorithm	OptOptimization
OptProblem	OptSimAnnealingAlgorithm
OptimizationLog	OptOptimizationsSet

FullDOEAlgorithm

Description

Type describing the Design of Experiment algorithm.

Inheritance path

Standard->Feature->OptFeature->OptGenericAlgorithm->OptGenericDOEAlgorithm
->FullDOEAlgorithm

Attributes

ConvergenceSpeed: Not appropriate

OptApproximationGradientAlgorithm

Description

Type describing the gradient algorithm with constraint.

Inheritance path

Standard->Feature->OptFeature->OptGenericAlgorithm->OptGenericOptimAlgorithm->OptApproximationGradientAlgorithm

Attributes

ConvergenceSpeed
N/A

OptConstraint

Description

Type describing an optimization constraint.

Inheritance path

Standard->Feature->OptFeature->OptGenericAlgorithm->OptGenericOptimAlgorithm->OptConstraint

Distance

Indicates the difference between the left hand side and the right hand side of the constraint. In optimization this right-hand side must be a constant.

Precision

This attribute is available only for equality constraints. It enables the user to specify a tolerance around the right-hand side value of the constraint. If distance (see attribute "Distance") is below this tolerance, the constraint is considered as satisfied.

Priority

N/A

Satisfaction

Indicates if the constraint is satisfied or not.

OptConstraintSatisfaction

Description

Type describing an optimization constraint.

Inheritance path

Standard->Feature->OptFeature->OptGenericAlgorithm->OptGenericOptimAlgorithm->OptConstraint

Distance

Indicates the difference between the left-hand side and the right-hand side of the constraint. In optimization this right-hand side must be a constant.

Precision

This attribute is available only for equality constraints. It enables the user to specify a tolerance around the right-hand side value of the constraint. If distance (see attribute "Distance") is below this tolerance, the constraint is considered as satisfied.

Priority

N/A

Satisfaction

Indicates if the constraint is satisfied or not.

OptFeature

Description

Father type of all optimization types.

Inheritance path

Standard->Feature->OptFeature

OptFreeParameter

Description

Type describing the free parameters available in the Product Engineering Optimizer workbench. Free parameters are parameters which vary according to the optimizer algorithm.

Inheritance path

Standard->Feature->OptFeature->OptFreeParameter

Attributes

HasRangesStep
Enables the user to check if the free parameter holds steps and/or ranges.

InfRange
Indicates the inferior range of the free parameter that can be indicated by the user in the Optimization editor.

Parameter
Corresponds to the underlying parameter of the free parameter. It also corresponds to the knowledgeware parameter of the model on which the optimization works.

Step
Indicates the steps of the free parameter that can be indicated by the user in the Optimization editor.

SupRange
Indicates the superior range of the free parameter that can be indicated by the user in the Optimization editor.

Value
Corresponds to the value of the free parameter.

OptGenericAlgorithm

Description

Corresponds to the base class of all the algorithms.

Inheritance path

Standard->Feature->OptFeature->OptGenericAlgorithm

Attributes

MaxTime
Corresponds to one of the termination criteria that can be set by the user in the Optimization dialog box.

MaxWoImprovement
Corresponds to one of the termination criteria that can be set by the user in the Optimization dialog box.

NbUpdatesMax
Corresponds to one of the termination criteria that can be set by the user in the Optimization dialog box.

StoppingCriterion
Enables the user to select one or several stopping criteria.

OptGenericDOEAlgorithm

Description

Corresponds to the base class of all the Design of Experiment algorithms.

Inheritance path

Standard->Feature->OptFeature->OptGenericAlgorithm->OptGenericDOEAlgorithm

OptGenericOptimAlgorithm

Description

Corresponds to the base class of all the optimization algorithms.

Inheritance path

Standard->Feature->OptFeature->OptGenericAlgorithm->OptGenericOptimAlgorithm

OptGoal

Description

Type describing the goal parameter of the optimization.

Inheritance path

Standard->Feature->OptFeature->OptGoal

Attributes

GoalComment
Enables the user to enter a comment.

GoalParameter
Enables the user to have access to the underlying knowledgeware parameter specified as the objective of the optimization.

GoalType
Enables the user to specify the type of goal for the optimization (minimum, maximum or target value.)

Precision
N/A

Priority
N/A

TargetValue
Enables the user to specify a target value to be reached by the goal parameter if the goal type is target value.

OptGradientAlgorithm

Description

Type describing the local optimization algorithm (gradient.)

Inheritance path

Standard->Feature->OptFeature->OptGenericAlgorithm->OptGenericOptimAlgorithm->OptGradientAlgorithm

Attributes

ConvergenceSpeed
See the Product Engineering Optimizer User's Guide.

OptOptimization

Description

Type describing the class that encapsulates the full description of the optimization (the algorithm, the problem, and the free parameters.)

Inheritance path

Standard->Feature->OptFeature->OptOptimization

Attributes

Algorithm
Algorithm used to solve the problem.

FreeParameters
List of the free parameters.

Problem
Problem to solve (Goal, and Constraints.)

UpdateVisualization
Enables the user to ask for visual update during optimization.

OptOptimizationsSet

Description

Type describing the collection of optimizations.

Inheritance path

Standard->Feature->OptFeature->OptOptimizationsSet

Attributes

Optimizations

List of all the optimizations in the set.

OptProblem

Description

Type describing the problem to be solved.

Inheritance path

Standard->Feature->OptFeature->OptProblem

Attributes

Constraints
List of the constraints (if any.)

Goals
List of the goals (if any.)

ProblemComment
Comment.

OptSimAnnealingAlgorithm

Description

Type describing the global optimization algorithm (Simulated Annealing.)

Inheritance path

Standard->Feature->OptFeature->OptGenericAlgorithm->OptGenericOptimAlgorithm->OptSimAnnealingAlgorithm

Attributes

ConvergenceSpeed
See the Product Engineering Optimizer User's Guide .

OptimizationLog

Description

Type describing the tool used to analyze the optimization results.

Inheritance path

Standard->Feature->OptFeature->OptimizationLog

Attributes

BestParm
Describes the parameter used to store the current value of the best result.

IndexOfBestSolInDT
Describes the list that can be used to save lines of the computations log.

NbEvalParm
Indicates the evaluation number.

PointsDT
Describes the optimization computations log.

Part Design

Chamfer	CounterboredHole	CounterdrilledHole
CountersunkHole	Draft	Groove
Hole	Pad	Pocket
Rib	Shaft	Shell
Slot	Stiffener	TaperedHole
Thickness	ThickSurface	Thread

Chamfer

Description

Describes a chamfer.

Inheritance path

Standard - Feature -> Standard - Visualizable -> MechanicalModeler -> GeometricFeature ->MechanicalModeler - MechanicalFeature

Attributes

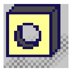
Angle
Defines the chamfer angle value

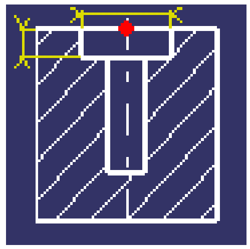
Length1
Defines the length from the selected edge on the first face.

Length2
Defines the length from the selected edge on the second face .

CounterboredHole

Description

Describes the mechanical feature of Hole type you create when you click the  icon in the Part Design workbench. For more information, refer to the *Part Design User's Guide*.



Inheritance path

Standard - Feature -> Standard - Visualizable -> MechanicalModeler - GeometricFeature -> MechanicaModeler - Body -> MechanicalModeler -> MechanicalFeature -> PartDesign - Hole

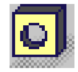
Attributes

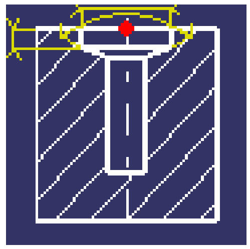
CounterboreDepth
Defines the counterbore depth

CounterboreDiameter
Defines the counterbore diameter

CounterdrilledHole

Description

Describes the mechanical feature of Hole type you create when you click the  icon in the Part Design workbench. For more information, refer to the *Part Design User's Guide*.



Inheritance path

Standard - Feature -> Standard - Visualizable -> MechanicalModeler - GeometricFeature -> MechanicaModeler - Body -> MechanicalModeler - MechanicalFeature -> PartDesign - Hole

Attributes

CounterdrillAngle
Defines the counterdrilled hole angle

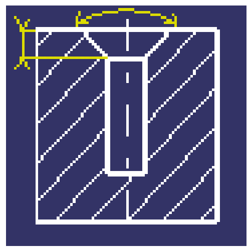
CounterdrillDepth
Defines the counterdrilled hole depth

CounterdrillDiameter
Defines the counterdrilled hole diameter

CountersunkHole

Description

Describes the mechanical feature of Hole type you create when you click the icon in the Part Design workbench. For more information, refer o the *Part Design User's Guide*.



Inheritance path

Standard - Feature -> Standard - Visualizable -> MechanicalModeler - GeometricFeature -> MechanicaModeler - Body -> MechanicalModeler - MechanicalFeature -> PartDesign - Hole

Attributes

CountersinkAngle
Defines the countersink angle

CountersinkDepth
Defines the countersink depth

CountersinkDiameter
Defines the countersink diameter

Draft

Description

Describes a draft.

Inheritance path

Standard - Feature -> Standard - Visualizable -> -> MechanicalModeler -> GeometricFeature ->MechanicalModeler - MechanicalFeature

Attributes

Angle
Defines the draft angle value

Groove

Description

Describes a groove.

Inheritance path

Standard - Feature -> Standard - Visualizable -> MechanicalModeler -> GeometricFeature ->MechanicalModeler - MechanicalFeature

Attributes

EndAngle
Defines the angle value for one direction

MergeEnd
Defines how the pad is trimmed to existing material

NeutralFiber
Defines how material is equally added to both sides of the profile

StartAngle
Defines the angle value for the second direction

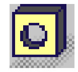
Thickness1
Defines the first thickness value

Thickness2
Defines the second thickness value

ThinMode
Defines if thickness is added

Hole

Description

Describes the mechanical feature of Hole type you create when you click the  icon in the Part Design workbench. For more information, refer o the *Part Design User's Guide*.

Inheritance path

Standard - Feature -> Standard - Visualizable -> MechanicalModeler - MechanicalFeature

Attributes

BottomAngle
Defines the bottom angle

BottomType
Defines the hole bottom type (v-bottom or flat)

Depth
Defines the hole depth

Diameter
Defines the hole diameter

DiameterThread
Defines the hole thread

HoleType
Defines the hole type (Simple, Tapered, Counterbored, Countersunk, Counterdrilled)

LimitType
Defines the hole limit (Blind, Up to Next, Up to Last, Up to Plane, Up to Surface)

Pitch
Defines the distance between each crest of the thread

TapSide
Defines the side or the thread (right or left)

Threaded
Defines the hole as threaded

ThreadingDepth
Defines the thread depth

Example

```
(for all) H:Hole
/*Displays the Hole activity and diameter */
Message("# activity: # - diameter: #", H->Name(), H.Activity, H.Diameter)
```

Pad

Description

Describes a pad.

Inheritance path

Standard - Feature -> Standard - Visualizable -> MechanicalModeler -> GeometricFeature ->MechanicalModeler - MechanicalFeature

Attributes

FirstLength
Defines the first length value

MergeEnd
Defines how the pad is trimmed to existing material

NeutralFiber
Defines how material is equally added to both sides of the profile

SecondLength
Defines the second length value

Thickness1
Defines the first thickness value

Thickness2
Defines the second thickness value

ThinMode
Defines if thickness is added

Pocket

Description

Describes a pocket.

Inheritance path

Standard - Feature -> Standard - Visualizable -> -> MechanicalModeler -> GeometricFeature ->MechanicalModeler - MechanicalFeature

Attributes

FirstLength
Defines the first length value

MergeEnd
Defines how the pad is trimmed to existing material

NeutralFiber
Defines how material is equally added to both sides of the profile

SecondLength
Defines the second length value

Thickness1
Defines the first thickness value

Thickness2
Defines the second thickness value

ThinMode
Defines if thickness is added

Rib

Description

Describes a rib.

Inheritance path

Standard - Feature -> Standard - Visualizable -> MechanicalModeler -> GeometricFeature ->MechanicalModeler - MechanicalFeature

Attributes

MergeEnd
Defines how the pad is trimmed to existing material

NeutralFiber
Defines how material is equally added to both sides of the profile

Thickness1
Defines the first thickness value

Thickness2
Defines the second thickness value

ThinMode
Defines if thickness is added

Shaft

Description

Describes a shaft.

Inheritance path

Standard - Feature -> Standard - Visualizable -> MechanicalModeler -> GeometricFeature ->MechanicalModeler - MechanicalFeature

Attributes

EndAngle
Defines the angle value for one direction

MergeEnd
Defines how the pad is trimmed to existing material

NeutralFiber
Defines how material is equally added to both sides of the profile

StartAngle
Defines the angle value for the second direction

Thickness1
Defines the first thickness value

Thickness2
Defines the second thickness value

ThinMode
Defines if thickness is added

Shell

Description

Describes a shell.

Inheritance path

Standard - Feature -> Standard - Visualizable -> MechanicalModeler -> GeometricFeature ->MechanicalModeler - MechanicalFeature

Attributes

DefaultInsideThickness
Defines the inside thickness value

DefaultOutsideThickness
Defines the outside thickness value

Slot

Description

Describes a slot.

Inheritance path

Standard - Feature -> Standard - Visualizable -> MechanicalModeler -> GeometricFeature ->MechanicalModeler - MechanicalFeature

Attributes

MergeEnd
Defines how the pad is trimmed to existing material

NeutralFiber
Defines how material is equally added to both sides of the profile

Thickness1
Defines the first thickness value

Thickness2
Defines the second thickness value

ThinMode
Defines if thickness is added

Stiffener

Description

Describes a stiffener.

Inheritance path

Standard - Feature -> Standard - Visualizable -> MechanicalModeler -> GeometricFeature ->MechanicalModeler - MechanicalFeature

Attributes

NeutralFiber
Defines how material is equally added to both sides of the profile

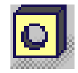
StiffenerMode
Defines the stiffener creation mode

Thickness1
Defines the first thickness value

Thickness2
Defines the second thickness value

TaperedHole

Description

Describes the mechanical feature of Hole type you create when you click the  icon in the Part Design workbench. For more information, refer o the *Part Design User's Guide*.

Inheritance path

Standard - Feature -> Standard - Visualizable -> MechanicalModeler - GeometricFeature -> MechanicalModeler - MechanicalFeature -> PartDesign - Hole

Attributes

TaperAngle
Defines the taper angle

Thickness

Description

Describes a thickness.

Inheritance path

Standard - Feature -> Standard - Visualizable -> MechanicalModeler -> GeometricFeature ->MechanicalModeler - MechanicalFeature

Attributes

DefaultThickness
Defines the default thickness value.

ThickSurface

Description

Describes a thicksurface.

Inheritance path

Standard - Feature -> Standard - Visualizable -> MechanicalModeler -> GeometricFeature ->MechanicalModeler - MechanicalFeature

Attributes

BotOffset
Defines the first offset value.

TopOffset
Defines the second offset value.

Thread

Description

Describes a thread.

Inheritance path

Standard - Feature -> Standard - Visualizable ->MechanicalModeler -> GeometricFeature ->MechanicalModeler -> MechanicalFeature

Attributes

Depth
Defines the thread depth value.

Diameter
Defines the thread diameter value.

Pitch
Defines the distance between each crest.

ThreadSide
Defines the side of the thread (right or left)

PartShared Package

ConstantEdgeFillet

Description

Describes the feature you create when you click the  icon in the Part Design workbench. For more information, please refer to the *Part Design User's Guide*.

Inheritance path

Standard - Feature -> Standard - Visualizable -> MechanicalModeler - GeometricFeature -> MechanicaModeler - Body -> MechanicalModeler - MechanicalFeature

Attributes

Radius
Defines the edge fillet radius.

UserPattern

Description

Describes a draft.
Inheritance path: Standard - Feature -> Standard - Visualizable -> MechanicalModeler -> GeometricFeature >MechanicalModeler -> MechanicalFeature -> PartSharedPackage-> Pattern

Attributes

LocationElement
Defines how the element is located.

RectPattern

Description

Describes a draft.

Inheritance path

Standard - Feature -> Standard - Visualizable -> MechanicalModeler -> GeometricFeature >MechanicalModeler -> MechanicalFeature -> PartSharedPackage-> Pattern -

Attributes

Step1
Defines the distance between instances in the first direction.

Step2
Defines the distance between instances in the second direction.

Product

Description

Describes a product.

Attributes

The attributes you can manipulate on this object are the properties you access in the Product tab of the Properties command. The attributes and methods below are only meaningful in the context of a CATProduct document.

PartNumber
Revision
Definition
Description
Nomenclature

Method(s)

Inherits all the MechanicalFeature methods.

Example

(for all) P:Product

Ship Structure Detail Design

CATSddBeaming

Description

Describes a beaming system.

Attributes

Inherits all ProductPackage attributes.

Methods

Inherits all ProductPackage methods.

Examples

CATSddBeaming:\PartNumber=="Product1"

CATSddInserting

Description

Describes an inserting system.

Attributes

Inherits all ProductPackage attributes.

Methods

Inherits all ProductPackage methods.

Examples

CATSddInserting:\PartNumber=="Product1"

CATSddOpening

Description

Describes an opening system.

Attributes

Inherits all ProductPackage attributes.

Methods

Inherits all ProductPackage methods.

Examples

CATSddOpening:\PartNumber=="Product1"

CATSddPlating

Description

Describes a plating system.
The following objects listed in the browser derive from this main object: DeckPlate, ShellPlate.
These objects are managed in the sample feature dictionary delivered with the product.

Attributes

Inherits all ProductPackage attributes.

Methods

Inherits all ProductPackage methods.

Examples

CATSddPlating:\PartNumber=="Product1"

CATSddStiffening

Description

Describes a stiffening system.
The following objects listed in the browser derive from this main object: DeckStiff, LongBulkhdStiff, ShellStiff, TransBulkhdStiff.
These objects are managed in the sample feature dictionary delivered with the product.

Attributes

Inherits all ProductPackage attributes.

Methods

Inherits all ProductPackage methods.

Examples

CATSddStiffening:\PartNumber=="Product1"

CATSddStiffeningOnFreeEdge

Description

Describes a stiffening system on a free edge.

Attributes

Inherits all ProductPackage attributes.

Methods

Inherits all ProductPackage methods.

Examples

CATSddStiffeningOnFreeEdge:\PartNumber=="Product1"

CATStrJointExt

Description

Describes a structure detail design connection.

Attributes

In addition to inheriting all ProductPackage attributes, specific attributes you can manipulate on this object are:

AddedPieces
Efficiency
Type
JoinedComponentName
NbJoinedComponents

Methods

Inherits all ProductPackage methods.

Examples

CATStrJointExt:\JoinedComponentName=="FunPlate1, FunPlate2"
/* Returns all objects connected to a connection */

Structure Preliminary Layout

CATSPLBoundedZone

CATSPLMoldedForm

CATSPLWrappingSurf

CATSPLBooleanOperator

CATSPLBoundedZone

Description

Describes a bounded zone.

The following objects listed in the browser derive from this main object: AuxElectricalRoom, AuxMachineryRoom, BerthRoom, EngineRoom,

Attributes

Inherits all ProductPackage attributes.

Methods

In addition to inheriting all ProductPackage methods, specific methods are:

SetColor(Red:Integer,Green:Integer,Blue:Integer) : Boolean
Assigns the color value specified. Enter RGB values to set color. Returns a 1 or 0 indicating whether or not the method was successful.

UpdateIDNaming() : Boolean
Forces ID naming rule update. Returns a 1 or 0 indicating whether or not the method was successful.

GetWall1() : Feature
Returns the molded form used to define the bounded zone along the X+ axis.

GetWall2() : Feature
Returns the molded form used to define the bounded zone along the Y+ axis.

GetWall3() : Feature
Returns the molded form used to define the bounded zone along the X- axis.

GetWall4() : Feature
Returns the molded form used to define the bounded zone along the Y- axis.

GetCeilling() : Feature
Returns the molded form used to define the bounded zone along the Z+ axis.

GetFloor() : Feature
Returns the molded form used to define the bounded zone along the Z- axis.

Example

```
P:CATSPLBoundedZone
P -> SetColor(255,0,0)
/* Changes the color of the current bounded zone to red */
```

CATSPLMoldedForm

Description

Describes a molded form.

The following objects listed in the browser derive from this main object: Deck, ExternalAppendage, LongBlk, TransBlk, Unspec,

Attributes

Inherits all ProductPackage attributes.

Methods

In addition to inheriting all ProductPackage methods, specific methods are:

SetColor(Red:Integer,Green:Integer,Blue:Integer) : Boolean
Assigns the color value specified. Enter RGB values to set color. Returns a 1 or 0 indicating whether or not the method was successful.

UpdateIDNaming() : Boolean
Forces ID naming rule update. Returns a 1 or 0 indicating whether or not the method was successful.

GetRefPlane() : Feature
Returns the reference plane used to define the molded form.

Example

```
P:CATSPLMoldedForm
P -> SetColor(255,0,0)
/* Changes the color of the current molded form to red */
```

CATSPLWrappingSurf

Description

Describes a wrapping surface.

The following objects listed in the browser derive from this main object: DeckHouse, ExternalHullForm, InternalHullForm, Sponson. These objects are managed in the sample feature dictionary delivered with the product.

Attributes

Inherits all ProductPackage attributes.

Methods

In addition to inheriting all ProductPackage methods, specific methods are:

SetColor(Red:Integer,Green:Integer,Blue:Integer) : Boolean
Assigns the color value specified. Enter RGB values to set color. Returns a 1 or 0 indicating whether or not the method was successful.

UpdateIDNaming() : Boolean
Forces ID naming rule update. Returns a 1 or 0 indicating whether or not the method was successful.

Example

```
P:CATSPLWrappingSurf
P -> SetColor(255,0,0)
/* Changes the color of the current wrapping surface to red */
```

CATSPLBooleanOperator

Description

Describes a composite volume.

Attributes

Inherits all ProductPackage attributes.

Methods

In addition to inheriting all ProductPackage methods, specific methods are:

SetColor(Red:Integer,Green:Integer,Blue:Integer) : Boolean
Assigns the color value specified. Enter RGB values to set color. Returns a 1 or 0 indicating whether or not the method was successful.

UpdateIDNaming() : Boolean
Forces ID naming rule update. Returns a 1 or 0 indicating whether or not the method was successful.

Example

```
P:CATSPLBooleanOperator
P -> SetColor(255,0,0)
/* Changes the color of the current bounded zone to red */
```

Structure Functional Design

CatStrPanelSystem

Description

Describes a panel system.
The following objects listed in the browser derive from this main object: DeckPanelSyst, ExtAppPanelSyst, LongPanelSyst, TransPanelSyst, UnspecPanelSyst, WDeckPanelSyst.

These objects are managed in the sample feature dictionary delivered with the product.

Attributes

Inherits all ProductPackage attributes.

Method(s)

Inherits all ProductPackage methods.

Example

```
CATStrPanelSystem:\PartNumber=="Product1"
```

CatStrPlateSystem

Description

Describes a plate system.
The following objects listed in the browser derive from this main object: DeckPlateSyst, ExtAppPlateSyst, LongPlateSyst, TransPlateSyst, UnspecPlateSyst, WDeckPlateSyst.

These objects are managed in the sample feature dictionary delivered with the product.

Attributes

Inherits all ProductPackage attributes.

Method(s)

Inherits all ProductPackage methods.

Example

CATStrPlateSystem:\PartNumber=="Product1"

CATStrFunInsertPlate

Description

Describes a functional insert plate.

Attributes

In addition to inheriting all ProductPackage attributes, specific attributes you can manipulate on this object are:

Material
Thickness
Weight

Methods

In addition to inheriting all ProductPackage methods, specific methods are:

SetColor(Red:Integer,Green:Integer,Blue:Integer)
Assigns the color value specified. Enter RGB values to set color.

Examples

```
CATStrFunInsertPlate:\Thickness==10mm
/* Returns all insert plates having a thickness of 10mm */
```

```
P:CATStrFunInsertPlate
if (P:\Thickness==10mm)
P -> SetColor(255,0,0)
/* Changes the color of all 10mm thick insert plates to red */
```

CATStrFunPillar

Description

Describes a functional pillar.

Attributes

In addition to inheriting all ProductPackage attributes, specific attributes you can manipulate on this object are:

CatalogName
FamilyName
Length
Material
ProfileType:

Methods

In addition to inheriting all ProductPackage methods, specific methods are:

SetColor(Red:Integer,Green:Integer,Blue:Integer)
Assigns the color value specified. Enter RGB values to set color.

Examples

```
CATStrFunPillar:\SectionName=="PIPE_SW_0.5"
/* Returns all pillars of section PIPE_SW_0.5 */
```

```
P:CATStrFunPillar
if (P:\SectionName=="PIPE_SW_0.5")
P -> SetColor(255,0,0)
/* Changes the color of all pillars of section PIPE_SW_0.5 to red */
```

CATStrFunStiffener

Description

Describes a functional stiffener.

Attributes

In addition to inheriting all ProductPackage attributes, specific attributes you can manipulate on this object are:

CatalogName
FamilyName
Length
Material
ProfileType:

Methods

In addition to inheriting all ProductPackage methods, specific methods are:

SetColor(Red:Integer,Green:Integer,Blue:Integer)
Assigns the color value specified. Enter RGB values to set color.

Examples

```
CATStrFunStiffener:\SectionName=="W14X30"
/* Returns all stiffeners of section W14X30 */
```

```
P:CATStrFunStiffener
if (P:\SectionName=="W14X30")
P -> SetColor(255,0,0)
/* Changes the color of all stiffeners of section W14X13 to red */
```

CATStrFunPlate

Description

Describes a functional plate.

Attributes

In addition to inheriting all ProductPackage attributes, specific attributes you can manipulate on this object are:

Material
Thickness
Weight

Methods

In addition to inheriting all ProductPackage methods, specific methods are:

SetColor(Red:Integer,Green:Integer,Blue:Integer)
Assigns the color value specified. Enter RGB values to set color.

Examples

```
CATStrFunPlate:\Thickness==10mm
/* Returns all functional plates having a thickness of 10mm */
```

```
P:CATStrFunPlate
if (P:\Thickness==10mm)
P -> SetColor(255,0,0)
/* Changes the color of all 10mm thick functional plates to red */
```

CATStrFunOpening

Description

Describes a functional opening.

Attributes

Inherits all ProductPackage attributes.

Method(s)

Inherits all ProductPackage methods.

Example

CATStrFunOpening:\PartNumber=="Product1"

CatStrStiffenerSystem

Description

Describes a stiffener system.
The following objects listed in the browser derive from this main object: DeckStiffSyst, ExtAppStiffSyst, LongStiffSyst, TransStiffSyst, UnspecStiffSyst, WDeckStiffSyst.

These objects are managed in the sample feature dictionary delivered with the product.

Attributes

Inherits all ProductPackage attributes.

Method(s)

Inherits all ProductPackage methods.

Example

```
CATStrStiffenerSystem:\PartNumber=="Product1"
```

CATStrFMFSkeleton

Exposed to permit interference checking on panel systems.

Standard

Visualizable Type	List
	Feature

Visualizable Type

Color Attribute

Allows you to get and set the color of a Generative Shape Design feature. The Color attribute is defined as a character string. When comparing values, bear in mind that the color attribute should be written in lower case and not in upper case. The color can be specified either by its full name or by its hexadecimal value:

black = "#000000";	white = "#FFFFFF";	navy = "#000080";	fushia = "#FF00FF";
green = "#008000";	marroon= "#800000";	blue = "#0000FF";	lime = "#00FF00";
silver = "#C0C0C0";	red = "#FF0000";	teal = "#008080";	olive = "#808000";
gray = "#808080";	purple= "#800080";	aqua = "#00FFFF";	yellow = "#FFFF00";

Show Attribute

Allows you to get/set the Show/NoShow mode of a Generative Shape Design feature. The Show/NoShow mode is to be set by a boolean (true/false).

Pick Attribute

Allows you to set the "Pickable" status of a feature so that it can or cannot not be selected in the geometrical area. The Pick attribute is to be set by a boolean.

Layer attribute

Allows you to get/set the layer associated with a feature.

Examples

```
Expert Rule1
(for all) G:GSMCurve
if G.Show == False
then G.Show = True
```

```
Expert Rule2
(for all) P:GSMPoint
P.Color = "blue"
P.Show = False
Message ("Curve pick is set to #", G.Pick)
P.Layer = 1
Message ("Point layer is set to #", P.Layer)
```

List

Description

List functions are used to manage lists of parameters, pads ... They enable the user to create lists, to add items to the list, to remove items from the list, to retrieve values from the list, to move elements of the list to another position, and to copy the content of a list into another one.

- **Size** ()
  Function used to return the number of items contained in the list.

- **AddItem** ()
  Function used to add an item to the list.

```
let list (List)
list.AddItem(PartBody\Hole.2 ,1)
list.AddItem(PartBody\Hole.3 ,2)
Message("#",list.Size())
```

- **Compute**()
  Function used to compute the result of an operation performed on the attributes supported by the features contained in the list.
  Example: List.1 .Compute("+","Hole","x.Diameter",Length.1)
  Where:

- List.1 is the name of the list on which the calculation will be performed
- + is the operator used. (Supported operators are: -, min, and max.)
- Hole is the type of the list items used for the calculation (to calculate the diameter, the type to be indicated is Hole, to calculate the volume, the type to be indicated is Solid)
- x stands for the list items. Note that the type of the items contained in the list should be identical.
- Length.1 is the output parameter.

- **GetItem** ()
  Function used to retrieve a value/item from the list

- **IsSorted** ()
  Limits the elements in the interface to instances of a certain type

- **RemoveItem** ()
  Function used to remove an item from the list.

- **ReorderItem** ( )
  Function used to move an element of the list to a new position.

- **Sum** ()
  Function used to copy the content of a list and paste it in another list.

Feature

Description

Describes the parent of all mechanical features.

Attributes

ID	Owner
Name	NamedURLs
UserInfoComment	

Methods

AbsoluteId Method	GetAttributeBoolean Method
GetAttributeInteger Method	GetAttributeReal Method
GetAttributeString Method	HasAttribute Method
Id Method	IsOwnedBy Method
IsSupporting Method	Name Method
Query	SetAttributeBoolean Method
SetAttributeInteger Method	SetAttributeReal Method
SetAttributeString Method	

Example

1. Create a part with several holes.
2. Add a real type parameter ("Real.1" for example) to one of the hole features. To do this, you must use the Knowledge Advisor product.
3. Create the rule below:

- List.1 is the name of the list on which the calculation will be performed.
- PartBody is the body on which the search will be carried out
- Hole is the Type.

- x.Diameter>50mm is the expression.

```
/* This rule resets the diameter of the hole */
/* which has "Real.1" as its parameter to the Real.1 value */
(for all) H:Hole
if H->HasAttribute("Real.1")
H.Diameter = 1mm*(H->GetAttributeReal("Real.1"))
```

You can use all the GetAttributexxx methods in that way.

- Add one or more drafts to the part.
- You can write the rule below:

```
(for all) Dr:Draft
/* Displays the names of the Drafts which have PartBody as their names */
```

AbsoluteId Method

Retrieves the path of a feature.

Syntax

feature.**AbsoluteId**(): String

Example

String.2=PartBody\Pad.1.Id() + PartBody\Pad.1.AbsoluteId()

Sample

KwrTopology.CATPart

GetAttributeBoolean Method

Returns the value of a boolean type parameter added to a given feature by using the Knowledge Advisor product. *parameterName* is the name of the boolean type parameter. It should be put between quotation marks (").This method enables to read:

- The attributes added to parameters using the Parameters Explorer.
- The real attributes added to objects.
- The User Properties of a product.

Syntax

feature.**GetAttributeBoolean(***String***):** Boolean

where the argument is name of the attribute.

Example

```
Message ("The value of the Boolean.1 attribute of # is #",
PartBody\Pad.1.Name(),
PartBody\Pad.1.GetAttributeBoolean("Boolean.1"))
```

Sample

KwrObject.CATPart

GetAttributeInteger Method

Returns the value of an integer type parameter added to a given feature by using the Knowledge Advisor product. *parameterName* is the name of the string type parameter. It should be put between quotation marks ("). This method enables to read:

- The attributes added to parameters using the Parameters Explorer.
- The real attributes added to objects.
- The User Properties of a product.

Syntax

feature.**GetAttributeInteger(***String***):** Integer

where *String* is name of the attribute. This name should be put between double-quotes.

Example

Integer.3=PartBody\Hole.1 .GetAttributeInteger("Integer.2")

Sample

KwrObject.CATPart

GetAttributeReal Method

Returns the value of a real or Length (in m) type parameter added to a given feature by using the Knowledge Advisor product. *parameterName* is the name of the string type parameter. It should be put between quotation marks ("). This method enables to read:

- The attributes added to parameters using the Parameters Explorer.
- The real attributes added to objects.
- The User Properties of a product.

Syntax

feature.**GetAttributeReal(***String***):** String

where *String* is name of the attribute. This name should be put between double-quotes.

GetAttributeString Method

Returns the value of a string type parameter added to a given feature by using the Knowledge Advisor product. *parameterName* is the name of the string type parameter. This method enables to read:

- The attributes added to parameters using the Parameters Explorer.
- The real attributes added to objects.
- The User Properties of a product.

Syntax

*feature***.GetAttributeString(***String***):** String

where *String* is name of the attribute. This name should be put between double-quotes.

Example

```
String.2 =PartBody\Pad.1 .GetAttributeString("String.1")
```

Sample

KwrObject.CATPart

HasAttribute Method

Determines whether the attribute specified in the argument belongs to the feature the method is applied to.

Syntax

feature.**HasAttribute(***String***):** Boolean

where *String* is name of the attribute. This name should be put between double-quotes.

Example

```
Boolean.2 =
PartBody\Hole.1.HasAttribute("Real.1")
```

Sample

KwrObject.CATPart

Id Method

Applies to a feature. Retrieves the identifier of a feature (not NLS).

Syntax

feature.**Id**(): String

Example

```
String.2=PartBody\Pad.1.Id() + PartBody\Pad.1.AbsoluteId()
```

Sample

KwrTopology.CATPart

IsSupporting

Function indicating if the object passed in argument is supported or not.

Example

```
H:Hole
H->IsSupporting("TaperedHole") == true
```

IsOwnedBy Method

Determines whether the feature specified in the argument is the parent of the feature the method is applied to. *featureName* should be put between quotation marks (").

Syntax

feature.**IsOwnedBy**(): Boolean

Example

Boolean.1=PartBody\Hole.1.IsOwnedBy(PartBody)

Sample

Topology.CATPart

Name Method

Applies to a feature. Retrieves the name of a feature. Cannot be used to rename a feature.

Syntax

feature.**Name**(): String

Example

```
String.1=PartBody\Pad.1.Name()
```

Sample

KwrTopology.CATPart

Query

Query()
Function used to search for the features located below the feature to which it applies and that verifies the specified expression and that adds these features to the list.
In the example below, the result of the search will return the holes of PartBody whose diameters are greater than 50mm.
Example: List.1=PartBody.Query("Hole","x.Diameter>50mm")
Where:

- List.1 is the name of the list on which the calculation will be performed.
- PartBody is the body on which the search will be carried out
- Hole is the Type of the searched feature.
- x.Diameter>50mm is the expression.

SetAttributeBoolean Method

Assigns the value specified in the second argument to the parameter whose name is specified in the first argument. *parameterName* is the name of the boolean type parameter whose value is to be modified. It should be put between quotation marks ("). *booleanvalue* is either TRUE or FALSE.

Syntax

feature.**SetAttributeBoolean(***String*, *Boolean***):** Void

where the first argument is name of the attribute while the second is the value to be assigned to it.

Example

```
if PartBody\Pad.1\Boolean.1 <> true
PartBody\Pad.1.SetAttributeBoolean("Boolean.1", true)
```

Sample

KwrObject.CATPart

SetAttributeInteger Method

Assigns the value specified in the second argument to the parameter whose name is specified in the first argument. *parameterName* is the name of the integer type parameter whose value is to be modified. *parameterName* should be put between quotation marks (").

Syntax

*feature***.SetAttributeInteger(***String*, *Integer***):** Void

where the first argument is name of the attribute while the second is the value to be assigned to it.

Example

```
if PartBody\Hole.1\Integer.1 <> 3
PartBody\Hole.1 .SetAttributeInteger("Integer.1", 3)
```

Sample

KwrObject.CATPart

SetAttributeReal Method

Assigns the value specified in the second argument to the parameter whose name is specified in the first argument. *parameterName* is the name of the real type parameter whose value is to be modified. *parameterName* should be put between quotation marks (").

Syntax

*feature***.SetAttributeReal(***String*, *Real***):** Void

where *String* is name of the attribute and *Real* the value to be assigned to the parameter.

Example

```
if PartBody\Hole.1\Real.1 <> 3
PartBody\Hole.1 .SetAttributeReal("Real.1",3)
```

Sample

KwrObject.CATPart

SetAttributeString Method

Assigns the value specified in the second argument to the parameter whose name is specified in the first argument. *parameterName* is the name of the string type parameter whose value is to be modified. *parameterName* and *stringvalue* should be put between quotation marks (").

Syntax

feature.**SetAttributeString(***String*, *String***):** Void

where the first argument is name of the attribute while the second is the value to be assigned to it.

Example

```
if PartBody\Pad.1.GetAttributeString("String.1") <> "String1"
PartBody\Pad.1 .SetAttributeString("String.1","This is a test")
```

Another syntax for the same rule is:

```
if PartBody\Pad.1\String.1 <> "String1"
PartBody\Pad.1.SetAttributeString("String.1","This is a test")
```

Sample

KwrObject.CATPart

Attributes

Id
Defines the feature identifier, i.e. the name primarily assigned to the feature at creation before any renaming has been done.

Owner
Defines the parent feature.

Name
Defines the feature name.

NamedURLs
Describes the URL that the user can add to a relation by clicking the Comment and URLs icon in the Knowledge Advisor workbench.

UserInfoComment
Describes the comment that the user can add in the Comment and URLs dialog box when adding a URL to a relation in the Knowledge Advisor workbench.

Topology

CATCell	CATEdge	CATFace
CATVertex	CATVolume	

CATCell

Description

Parent of the CATEdge, CATFace, CATVertex and CATVolume objects.

Inheritance path: Standard - Feature -> Standard - Visualizable.

CATEdge

Description

Describes an edge of a solid, that is any edge of a PartBody type feature. When manipulating such edges in expert rules and checks, note that most rounded edges (the edges of hole type features for example) are actually divided into sub-edges by vertices. A hole is made up of 6 edges delimiting two faces. Each circular edge is actually divided into two circular halves delimited by two diametrically opposed vertices and two linear edges join the vertices of either circular edges so that the resulting hole is made up of two half-cylindrical faces.

Inheritance path: Standard - Feature -> Standard - Visualizable ->Topology - CATCell

Example

Below is a rule example that can be added to any solid.

```
(for all) Ed:CATEdge
if Ed->length()==50mm
Message("All edges are 50mm long")
```

You can also write:

```
(for all) Ed:CATEdge
if length(Ed)==50mm
Message("All edges are 50mm long")
```

CATFace

Description

Describes a face of a solid, that is any face of a PartBody type feature. A CATFace object can be planar or not.

Inheritance path: Standard - Feature -> Standard - Visualizable ->Topology - CATCell

Example

Below is a check example that can be added to any solid.

(for all) Sur:CATFace
Sur->area()==100mm2

You can also write:

(for all) Sur:CATFace
area(Sur)==100mm2

CATVertex

Description

Describes a vertex of a solid, that is any point used as a reference to define a face or an edge of a PartBody type feature.

Inheritance path: Standard - Feature -> Standard - Visualizable ->Topology - CATCell

CATVolume

Description

Describes the volume of a solid.

TPSPackage

CATTPSAllAnnotations

CATTPSCapture

CATTPSFlagNote

CATTPSNonSemantic

CATTPSReferenceFrame

CATTPSSemantic

CATTPSSet

CATTPSView

CATTPSAllAnnotations

Enables the user to retrieve all the annotations contained in the document from an Expert Rule.

Syntax

'Functional Tolerancing & Annotations'.Annotation

Example

Retrieves all the annotations in the document.

```
'Functional Tolerancing & Annotations'.Annotation
```

CATTPSCapture

Enables the user to retrieve all the annotation captures contained in the document from an Expert Rule.

Syntax

'Functional Tolerancing & Annotations'.Capture

Example

Retrieves all the annotation captures in the document.

```
'Functional Tolerancing & Annotations'.Capture
```

CATTPSFlagNote

Description

Describes an annotation flagnote.
Inheritance path: Standard - Feature -> TPSPackage - CATTPSFlagNote

Attributes

HavingDocument

Defines the flagnote hyperlink reference.

LabelString

Defines the flagnote label.

Example

Retrieves flagnotes in the document.

```
'Functional Tolerancing & Annotations'.'Flag Note'
```

CATTPSNonSemantic

Description

Describes a non-semantic annotation.
Inheritance path: Standard - Feature -> TPSPackage - CATTPSNonSemantic

Attributes

ProductReference
Defines the product reference for the non-semantic annotation.

Example

Retrieves non-semantic annotation in the document.

```
'Functional Tolerancing & Annotations'.'Non Semantic Annotation'
```

CATTPSReferenceFrame

Description

Describes a datum reference frame.
Inheritance path: Standard - Feature -> TPSPackage - CATTPSReferenceFrame

Attributes

LabelRefFrame
Defines the reference frame label..

Example

Retrieves datum reference frame in the document.

```
'Functional Tolerancing & Annotations'.'Datum Reference Frame'
```

CATTPSSemantic

Description

Describes a semantic annotation.
Inheritance path: Standard - Feature -> TPSPackage - CATTPSSemantic

Attributes

AdvStatusSemanticAnnotation

Defines the advanced status for a semantic annotation: disconnected leader, invalid semantic, invalid tolerancing feature, unresolved, unresolvedsemantic, unresolvedtolerancing.

StatusSemanticAnnotation

Defines the status for a semantic annotation: valid or invalid.

Example

Retrieves semantic annotation in the document.

```
'Functional Tolerancing & Annotations'.'Semantic Annotation'
```

CATTPSSet

Description

Describes an annotation set.
Inheritance path: Standard - Feature -> TPSPackage - CATTPSSet

Attributes

ProductReference
Defines the product reference for the annotation set.

Example

Retrieves annotation set in the document.

```
'Functional Tolerancing & Annotations'.'Annotation Set'
```

CATTPSView

Description

Describes a semantic annotation.
Inheritance path: Standard - Feature -> TPSPackage - CATTPSReferenceFrame

Equipment Support Structure

StrFoundationExt

STRMember

STRPlate

StrFoundationExt

Description

Describes a structural member.

Attributes

Inherits all ProductPackage attributes.

Methods

Inherits all ProductPackage methods.

Example

```
P:StrFoundationExt
/* Returns all foundations */
```

STRMember

Description

Describes a structural member.

Attributes

In addition to inheriting all ProductPackage attributes, specific attributes you can manipulate on this object are:

CatalogName
FamilyName
Length
Material
ProfileType: section shape (beam, round, square, etc.)
SectionName
Weight

Methods

In addition to inheriting all ProductPackage methods, specific methods are:

SetColor(Red:Integer,Green:Integer,Blue:Integer)
Assigns the color value specified. Enter RGB values to set color.

Examples

```
STRMember:\SectionName=="W14X30"
/* Returns all members of section W14X30 */
```

```
P:STRMember
if (P:\SectionName=="W14X30")
P -> SetColor(255,0,0)
/* Changes the color of all members of section W14X13 to red */
```

STRPlate

Description

Describes a structural plate.

Attributes

In addition to inheriting all ProductPackage attributes, specific attributes you can manipulate on this object are:

Material
Thickness
Weight

Methods

In addition to inheriting all ProductPackage methods, specific methods are:

SetColor(Red:Integer,Green:Integer,Blue:Integer)
Assigns the color value specified. Enter RGB values to set color.

Examples

```
STRPlate:\Thickness==10mm
/* Returns all plates having a thickness of 10mm */
```

```
P:STRPlate
if (P:\Thickness==10mm)
P -> SetColor(255,0,0)
/* Changes the color of all 10mm thick plates to red */
```

Using the Check Analysis Tool

The Global Analysis Tool is designed to manage Expert and Advisor checks wherever they may be located in the specification tree. It helps end-users understand the validation status of their designs and allows navigation by checks or violations and highlights failed components. The user can:

- Access information concerning failing items,
- Gather information concerning objects and checks,
- Perform automatic corrections if need be.

The Global Analysis tool can be accessed at the session level by clicking the icon in the toolbar. This icon provides the user with a simple Checks status:

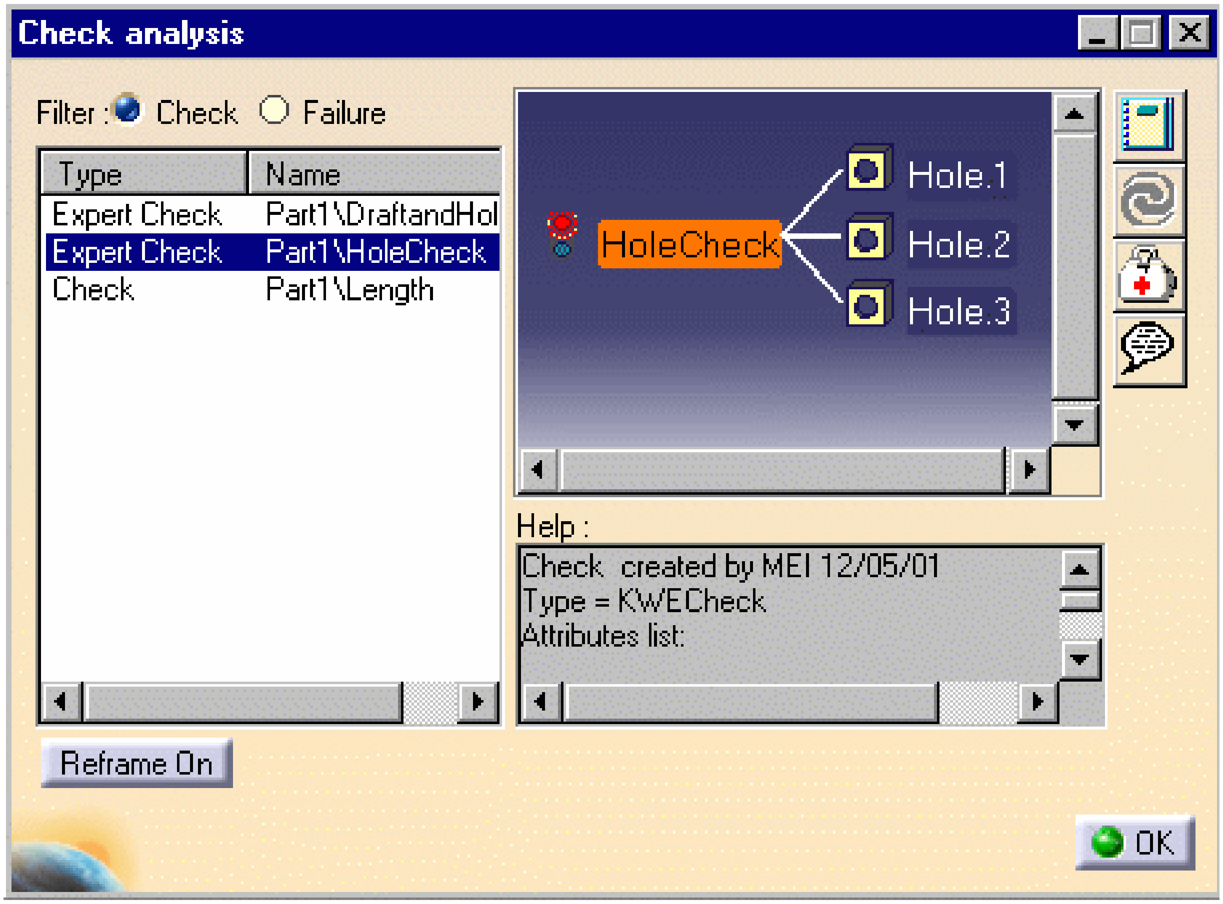
 All the checks are updated and could be fired successfully.

 The checks need to be updated.

 All the checks are updated and at least one of them is incorrect.

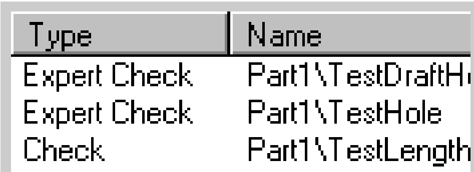
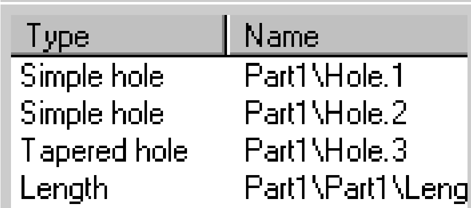
Check Analysis Tool Window

Click the  icon in the toolbar to access the Check analysis window



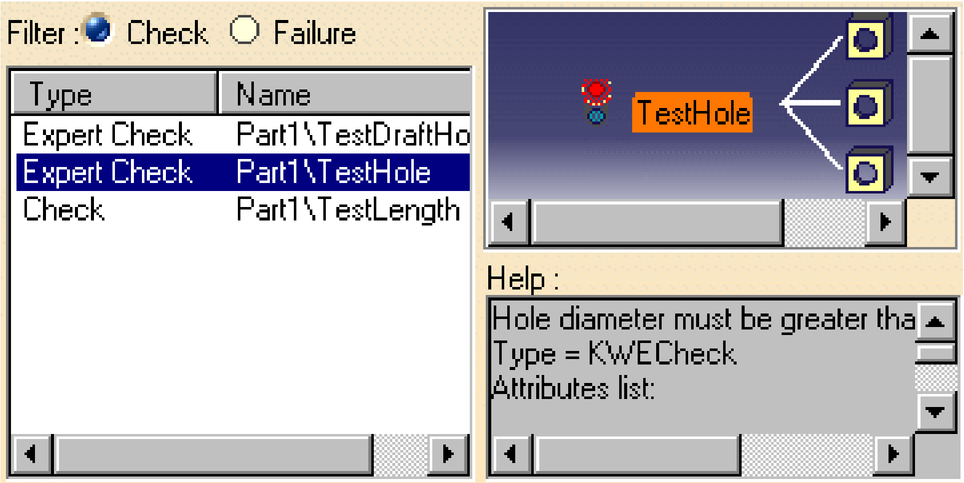
Filter section

This option enables the user to apply a filter to checks or to the items that failed.

<table>
<tr><td>Check</td><td>Only the Expert and Advisor checks that failed when updating the check report are displayed.</td><td>
</td></tr>
<tr><td>Failure</td><td>All the items that failed when updating the check report are displayed.</td><td>
</td></tr>
</table>

Help section

To display the help section associated with each item of the list, double-click the desired item. The following view is displayed:



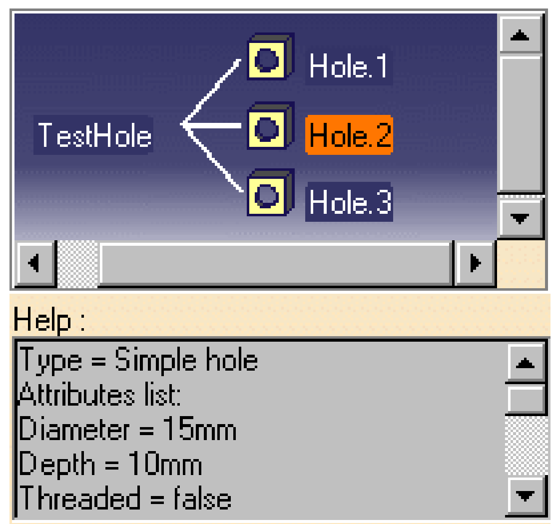
The check and the items it controls are displayed in the view as well as its current status.

The items entered when creating the check are displayed:

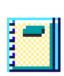
- Associated comments
- Type
- Attributes
- Variables
- Name
- Owner of the check...

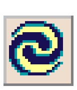
In the graphic above, the selected check is TestHole, it checks the holes of the CATPart file (3 of them do not pass the check because their diameters is not superior to 15mm), and the attributes are displayed corresponding to the data entered when creating the check.

Note that it is also possible to select the items associated to the check.
To do so, double-click the desired item in the view: The Help section shows the information concerning this item (see graphic opposite.)



Toolbar

 Click this icon to generate the customizable check report. To know more about the check report, see Customizing Check Reports.

 Click this icon to solve the checks created in your document.

 Click this icon to launch the correction method specified in the Check Editor when creating the check. To know more about the correction method, see Launching a Correction method and Using the Check Editor.

 Click here to display the URL associated to the object, or to assign an URL to an object. To know more, see the *Knowledge Advisor User's Guide*.

Customizing Check Reports

The reports generated by the Global Check Analysis Editor can be customized. You can choose to display a xml or a html report.

Displaying a HTML report

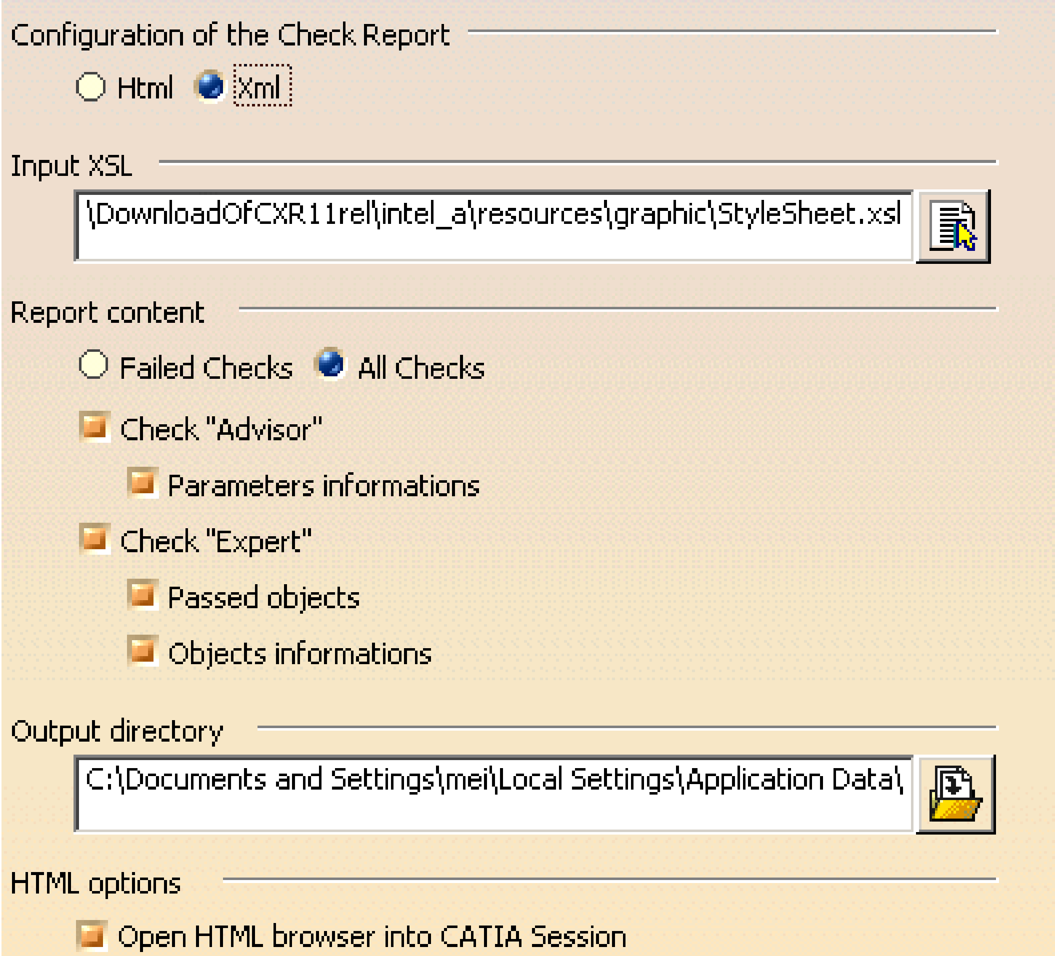
To generate a html report when performing the check analysis, go to **Tools->Options->General->Parameters and Measure,** and select the **Report generation** tab. Select **Html** in the **Configuration of the Check Report** area.

In this case, only the **Check Advisor**, the **Check expert** and the **Passed objects** options are available in the **Report content** area. You can specify the output directory containing the generated HTML report in the **Output directory** field.

Select **Html** if you use a Netscape browser.

Displaying a XML report

To display a XML report when performing the check analysis, go to **Tools->Options->General->Parameters and Measure** and select the **Report generation** tab. Select **Xml** in the **Configuration of the Check Report** area. . The following window opens:



The Report generation tab is made up of 4 different areas: The Input XSL, the Report Content, the Output directory, and the HTML options areas.

Input XSL area

This field enables the user to select the XSL style sheet that will be applied to the generated XML report. The StyleSheet.xsl file is the default XSL file, but you can use your own template.

Report content area

Failed Checks		If checked, the generated report will contain information about the failed checks only.
All Checks		If checked, the generated report will contain information about all the checks contained in the document.
Check advisor		If checked, the generated report will contain information about all the Knowledge Advisor checks contained in the document.
	Parameters information	If checked, the generated report will contain information about the parameters of the Advisor checks.

Check expert		If checked, the generated report will contain information about all the Knowledge Expert checks contained in the document.
	Passed objects	If checked, the generated report will contain information about the objects that passed Expert checks as well as information about the parameters of these objects (diameter, depth, pitch,...).
	Objects information	If checked, the generated report will contain information about all the objects contained in the Expert checks as well as information about the parameters of these objects (diameter, depth, pitch,...).

Output directory area

This field enables the user to select the output directory containing the generated XML report.

HTML options area

This option is available for Windows only. It enables the user to define if the report will be opened in a session (in this case, the check box should be checked) or if it will be opened in an Internet Explorer session (in this case, the check box should remain unchecked.)

Note that it is highly recommended not to use this report as a basis for macros or for other applications. It is only provided for information purposes.

Using Knowledge Expert Language (KWE)

From Version 5 Release 7, you can use a language close to VB called KWE when defining rules and checks. All tasks related to this new capability are updated accordingly.

Declaring Variables

The variables described below are those you can declare in the For All field of the Rule Editor or in the Check Editor.

Variables should comprise letters and/or digits.

- Variable names have no size limitation.
- Variable names are case-sensitive character strings.
- Variable names should not conflict with unit names. To get an exhaustive list of the units supported by Knowledge Expert, see the Units field of the Knowledge Expert browser in the Check/Rule Editor.
- Types starting with a digit (2DCircle) or containing a special character (+,-,...) should be written as follows: %2DCircle%, "%" acting like delimitators.

To declare variables in the *For all* field of the Check/Rule Editor, see Defining Types in the Check/Rule Editor.

When using KWE language, you may now declare autoreferencing variables in the Check or the Rule body by using Thisrule or Thischeck. To know more, see Accessing the Expert Check in the Check Body.

Using Types in the Check/Rule Editor

The types described below are those you can use in the Rule Editor or in the Check Editor by using the Object Browser.

In the (*For all*) field, use the following syntax: ***type_identifier*:type_*name***

where *type_identifier* corresponds to the variable you want to declare and *type_name* stands for the type to be declared and supported by Knowledge Expert (displayed in the middle window of the Object Browser).

Note that you can insert feature names (types) by keying them in the *For all* field, by selecting them in the Check/Rule Editor browser or by clicking the features in the geometry window or in the specification tree.

Examples:

H:Hole ; S:Shell
Hl:Hole ; Sel:Shell

Type declarations should be separated with semi-colons.

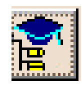
To know more about the Variables declaration in the *For all* field, see Declaring Variables.
To know more about the types supported by the Knowledge Expert Object Browser, see Using the Objects Library or the Knowledge Expert Browser in the application.

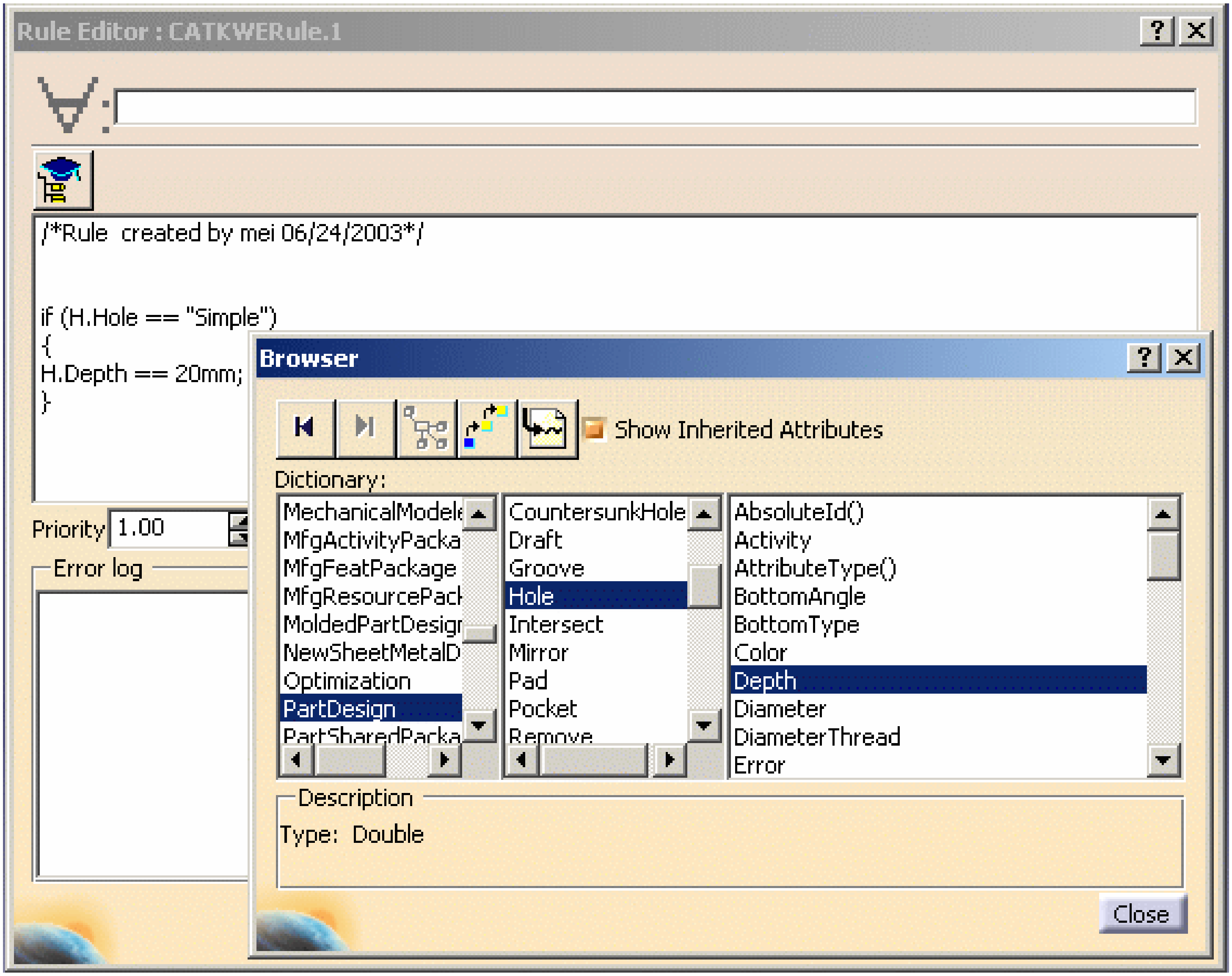
Using Types Attributes

The types attributes described below are those you can select in the Object Browser available from the Rule Editor or the Check Editor.

Types are allocated attributes that you can key in in the check or rule body or that you can select in the Object Browser.

- If you want to use the Object Browser, proceed as follows:

1. From the Knowledge Expert Workbench, access the Rule/Check Editor.
2. Click the  or the  icon, create a new rule or a new check, then click the  icon in the Rule or the Check Editor.
3. Click the package you want to work with in the left window (PartDesign in the graphic below), the type (Hole in the graphic below) and the attribute (Diameter in the graphic below), and click **Close**.



- If you want to manually key in the statement, use the following syntax:

type_identifier.object_name

Example:

if H.Diameter > 10.0 then H.Activity = False

Using Control Structures

Knowledge Expert Syntax

The control structure described below is the one the user can use in the Rule Editor or in the Check Editor.

Any KWE script is built out of a series of statements. A statement can be an assignment, a function call, or a conditional statement. Statements usually end with a semicolon. In addition, statements can be grouped into a statement-group by encapsulating a group of statements within curly braces.

If Construct

The `if` construct is the only one in the KWE language. It allows for conditional execution of code fragments. Note that "if" constructs cannot be nested.

```
if(Hole.HoleType == "Simple")
{
Hole.Diameter = 24mm;
}
```

- and and or are supported in the KWE statements.
- To execute a statement if a certain condition is met, and a different statement if the condition is not met, the user should create two different rules or checks. This is how to provide "else" functionality in other languages.

Knowledge Advisor Syntax

Note that when using this syntax, do not insert ; at the end of the instructions.

Conditional Statements

Rules

if ... else ... else if

Conditionally executes a group of statements, depending on the value of an expression. You can use either block form syntaxes:

if *condition statements* [**else** *elsestatements*] *or*

```
if condition
   { statements }
[else if condition-n
   [ { elseifstatements } ] ] . . .
[else
   [ { elsestatements } ] ]
```

You can use the single-line form (first syntax) for short, simple rules. However, the block form (second syntax) provides more structure and flexibility than the single-line form and is usually easier to read, maintain, and test.

The **else** and **else if** clauses are both optional. You can have as many **else if** statements as you want below a block **if**, but none can appear after the **else** clause. Block **if** statements can be nested that is, contained within one another.

Checks

statement1 **=>** *statement2* (**if** *statement1* **then** *statement2*)

Displays a message (if type is Warning or Information) and turns to red in the specification tree each time *statement2* is invalid as *statement1* is fulfilled.

Using Operators

Arithmetic operators

+ Addition operator (also concatenates strings)

- Subtraction operator

* Multiplication operator

/ Division operator

Comparison Operators

<> Not equal to

== Equal to

>= Greater or equal to

<= Less than or equal to

< Less than

> Greater than

Others

= Assignment operator

** Exponentiation operator

Filter Operator

Using the Filter Operator

The Filter Operator (**=>**) is designed for implication: It enables the user to restrict the check operation on a subset of the features that were specified in the **For all** field:

Example 1

Given the check below:

H:Hole H->HasAttribute("Cost")=>H.Diameter>10mm	The check report will only provide you with the results of the **H.Diameter > 10 mm** tests on the holes with a Cost attribute. Tests on other holes won't be performed.

Example 2

H.Diameter > 3mm =>H.Depth>10mm	If the diameter of a hole is greater than 3mm, its depth must be greater than 10mm.

Using Functions

Functions supported by Knowledge Expert may be entered in the check or rule body or can be selected from the Object Browser.

If you want to use the Object Browser, proceed as follows:

1. From the Knowledge Expert Workbench, access the Rule/Check Editor.
2. Click the or the icon, create a new rule or a new check, then click the icon in the Rule or the Check Editor.
3. Click the package you want to work with in the left window (PartDesign in the graphic below), the type (Hole in the graphic below) and the attribute (Diameter in the graphic below), and click **Close**.

- The *Show Inherited Attributes* box should be checked for the methods to be visible.
- The -> operator is required.

Example

```
(for all) Prod1:Product ; Prod2:Product

if Prod1 > Prod2 and ClashOrContact("Clash", Prod1,Prod2)
Message ("Products clashing are # and #", Prod1->Name(),Prod2->Name())
```

Using Constants

The following constants are specified or recognized by when programming rules and checks. As a result, they can be used anywhere in a relation in place of the actual values.

- false - one of the two values that a parameter of type Boolean can have
- true - one of the two values that a parameter of type Boolean can have
- PI - **3.14159265358979323846 -** The ratio of the circumference of a circle to its diameter.
- E - The base of natural logarithm - The constant *e* is approximately **2.718282**.

Advanced Tasks

Prior to performing the tasks listed below, read the Basic Tasks section which provides an overview of the functions that can be performed using the Knowledge Expert product.

You can also find useful information in Knowledge Expert Automation Principles (see the CAA documentation). The Knowledge Expert product does not provide you with journaling capabilities, but you can write macros replaying most of the Knowledge Expert operations.

Launching a Check Correction Method

This task explains how to create a check and use a VB correction method to make the invalid features fulfill the check.

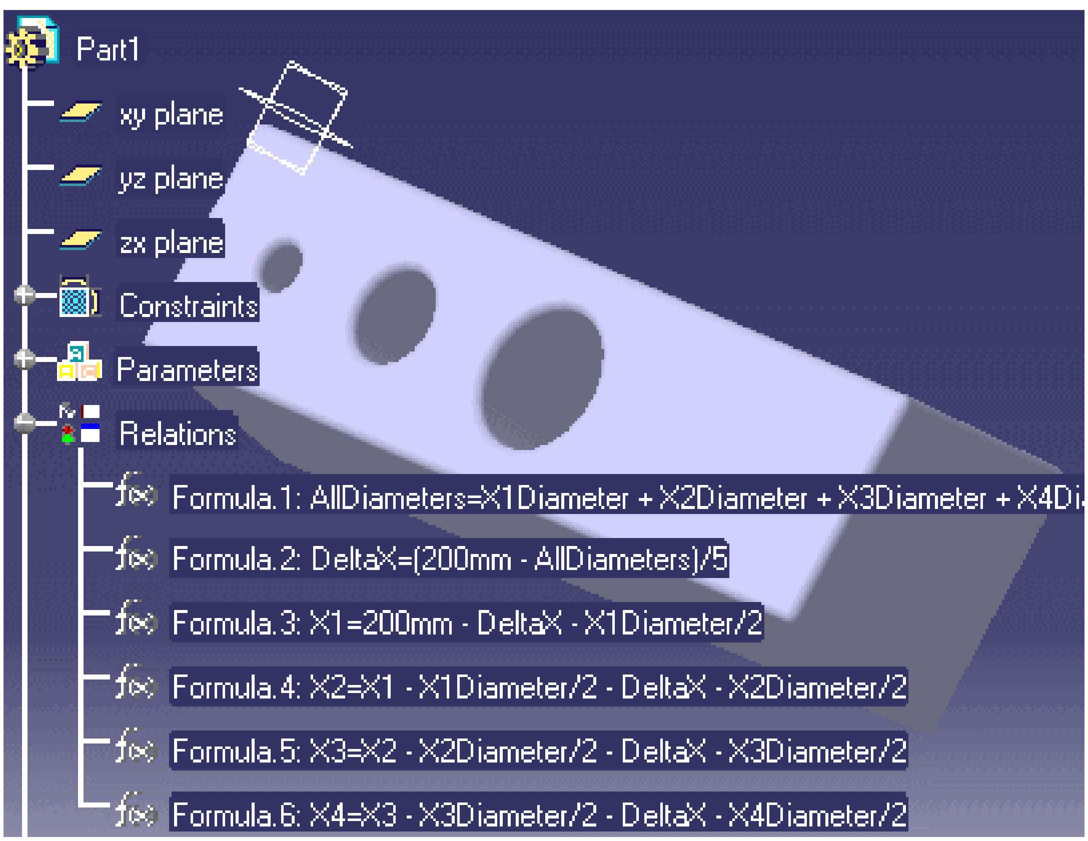
1. Open the KwxUseCase1.CATPart document.

   This document is a draft built from a rectangular pad. Four holes are evenly distributed along the draft length (200mm). The hole anchor point positions are driven by formulas. If a hole diameter is modified, the hole positions are recalculated so that the space between two successive holes remains constant along the draft length.

   The initial status of the Holes is:

   - Hole.1 - Deactivated - Diameter = 40mm
   - Hole.2 - Activated - Diameter = 30mm
   - Hole.3 - Activated - Diameter = 20mm
   - Hole.4 - Activated - Diameter = 10mm

   The following screen is displayed.



2. Access the Knowledge Expert workbench, then click the  icon. In the dialog box which is displayed, enter a check name and a comment, then click OK. The three check editor is displayed.

3. Select the Condition tab and enter the check below:

```
(for all) H:Hole
H.Activity == true
```

   Note that the syntax below is also valid:

```
(for all) H:Hole
H.Activity == true
```

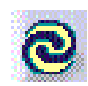
4. In the Correction tab, select VB Script as correction method and enter the script below into the edition box:

```
Dim oPart1 As Part
Set oPart1 = H.Parent.Parent.Parent.Parent

oPart1.Activate H
   oPart1.Update
```

5. Select the Report tab and enter the text below into the Help Message edition box:

```
Checks that all the document holes are activated
```

6. Click **OK** to add the check to the rule base, then click the  icon to solve the rule base. In the specification tree, the check icon is red. This indicates that not all the part holes are activated. Keep your document open and proceed to the next task.

Defining Rules working on UDFs

This task explains how to define expert rules working on User Defined Features.

To know more about UDFs, see the *Product Knowledge Template* documentation.

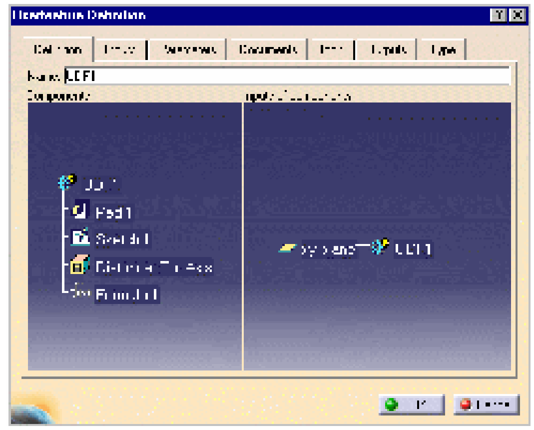
To perform this task, it is highly recommended to be familiar with the User Defined Features concept and with the Part Design workbench. To know more about the UDF concept and Part Design, see the *Product Knowledge Template* and the *Part Design* documentations.

Go to the **Tools->Options...->Parameters and Measure** menu, click the **Language** tab. In the **Reference Directory For Types**, enter the path of the .CATGscript containing the type that is to be generated and click **OK**.

1. From the **Start->Mechanical Design** menu, access the **Part Design** workbench.
2. Click the **Sketcher** icon, the xy plane, and create a rectangular sketch.
3. Create a parameter of **Length** type. To do so, proceed as follows:
   - Click the icon.
   - Select Length in the scrolling list to define the type of the parameter, click the **New parameter of type** button, change the name of the parameter (Distance_To_Axis in this scenario), and set its value to 0mm. Click **OK**.
4. Create a constraint on the Distance_To_Axis parameter that will define the distance between the left vertical edge of the sketch and the VDirection-AbsoluteAxis. To do so, proceed as follows:
   - Select the axis and the edge.
   - Click the **Constraint** icon () and click in the geometrical area to define the constraint.
5. Double-click the digital value of the constraint, right-click the Value field, select **Edit formula...**, and enter PartBody\Sketch.1\Offset.5\Offset = Distance_To_Axis by selecting Distance_To_Axis in the specification tree or in the **Member of All** window. Click **OK** twice.
6. Close the sketcher and create a pad by clicking the Pad icon ().
7. Create the user feature. To do so, proceed as follows:
   - Choose the **Insert->UserFeature->UserFeatureCreation...** command.

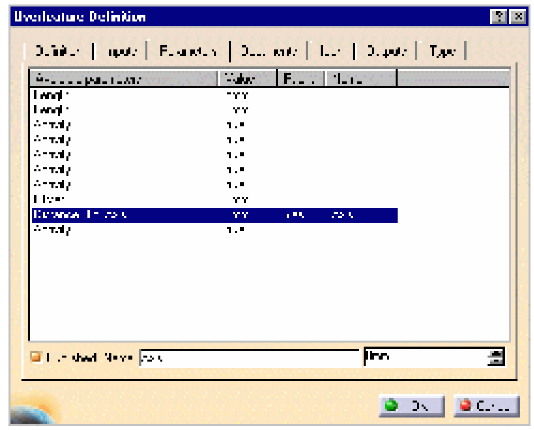
- Select the **Definition** tab, rename the user feature (UDF1 in this scenario), and add the pad, the sketch and the parameter by selecting them in the specification tree.

(Click the graphic opposite to enlarge it.)

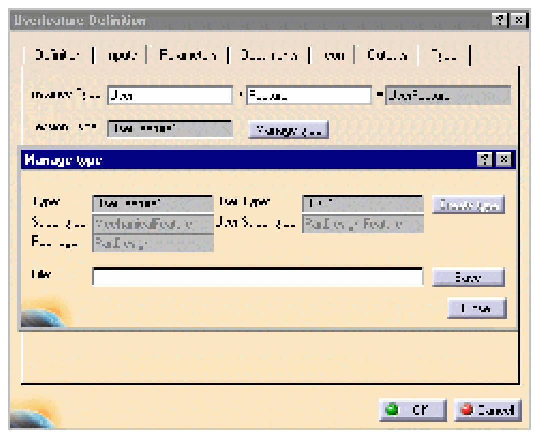


- Publish the **Distance_To_Axis** parameter. To do so, select the **Parameters** tab, select the Distance_To_Axis parameter in the **Available parameters** column to publish the Distance-To-Axis parameter.
  Check the **Published Name** check box, rename the parameter (Axis in this scenario).

(Click the graphic opposite to enlarge it.)

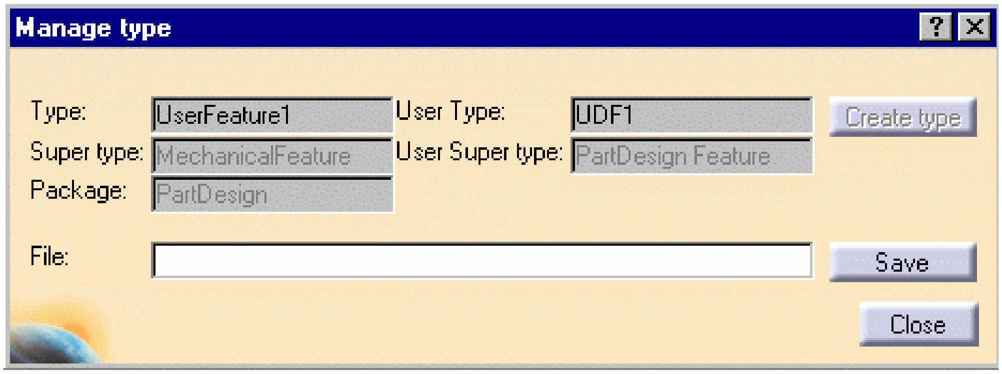


- Select the **Type** tab, and, in the **Instance Type** field, enter the name of the UDF type (UserFeature1 in this scenario).
  To do so, enter a prefix in the first **Instance Type** field. This prefix should be made of at least 3 characters. It will enable the user to gather UDFs of the same kind by using their names.
  In the second field, enter the identifier of the UDF. Hit the **Enter** key or press the tab key. The Instance type is created and the **Manage type** button is available.



Click the graphic above to enlarge it.

- Click the **Manage type** button: the **Manage Type** window opens.



Type: corresponds to the type you have created in the Instance type field.

User Type: corresponds to the type you have created in the Definition tab.

Super Type: corresponds to the type from which the type you are creating will inherit.

Package: corresponds to the workbench in which the type you are creating will be displayed.

The packages available here are GSM, MechanicalModeler and Part Design.

File: corresponds to the CATGscript file you can create and that you will be able to use in your next sessions.
- Click **Create type**, **Save**, and **Close** if you want to use the file in another session.
- Click **Create type**, and **Close** if you want to use the created type in the current session only.

8. For the purpose of this scenario, it is highly recommended to select the default settings. Click **Create type**, **Save**, and **Close**.
9. Click **OK** to close the Userfeature Definition window: a UDF is added to the UserFeatures node.
10. Add a new body to the CATPart file. To do so, access the Part Design workbench and go to **Insert->Body**. Rename the inserted body (Result_Body in this scenario).
11. Access the Generative Knowledge workbench.
12. Click the **Loop** icon, and the  icon located in the loop editor. Enter the following formula: **LastItemForAction=3**. Click **OK**.

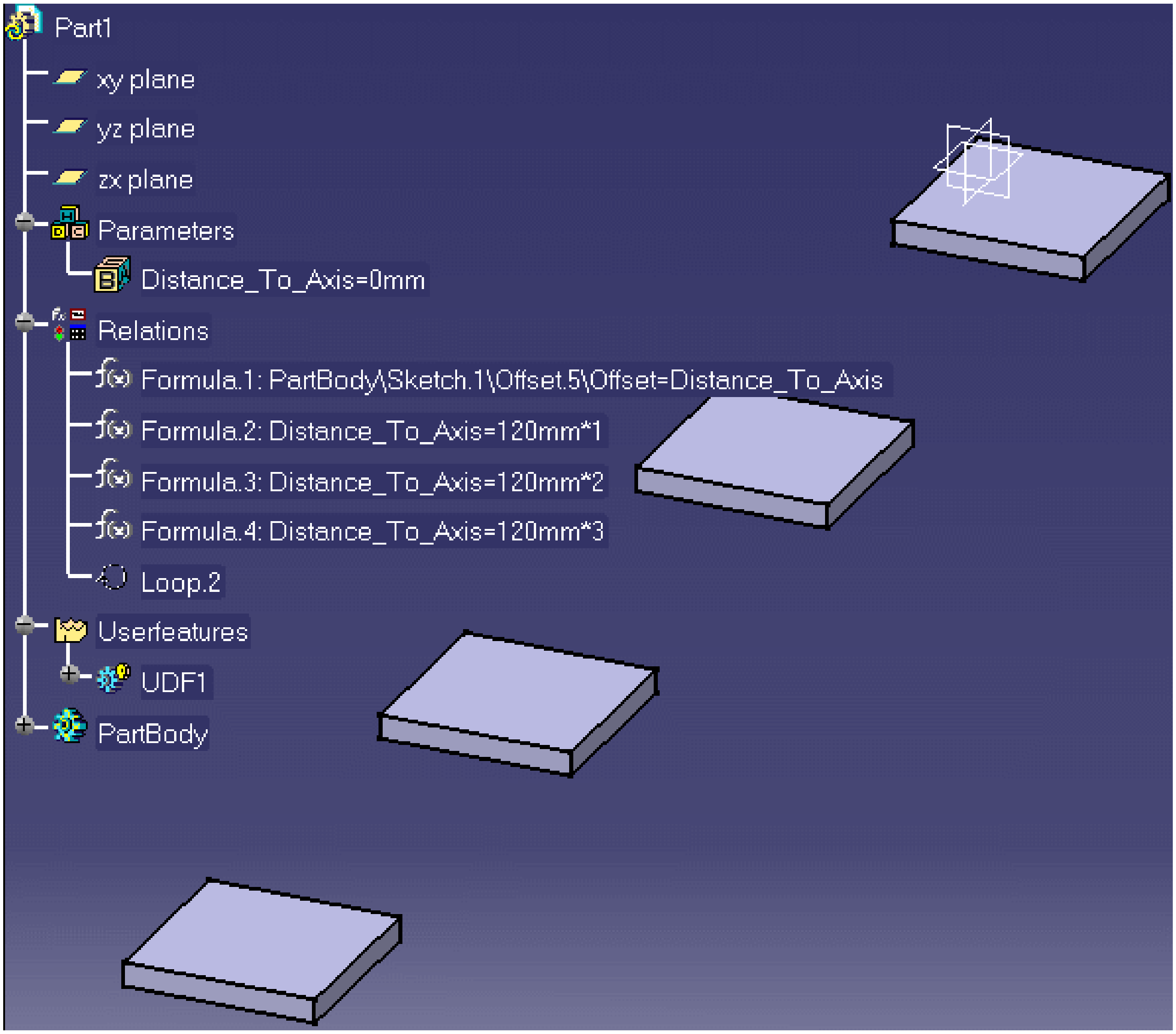
    The Loop function enables the user to instantiate a UDF in its creation CATPart.

13. Select **Result_Body** in the specification tree, and click the **Select Context** button.
14. In the Loop editor, enter the following script.

```
UDF_$i$ isa UDF1 // UDF1 is the name assigned to the UDF
{
Axis = 120mm * $i$; //Name of the published parameter
}
```

    In the script above, the UDF is instantiated by using the user type.

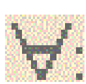
15. Click **OK**: the UDFs are instantiated in the CATPart. 4 pads are displayed: the one you created and the 3 instantiated ones.



16. Access the Knowledge Expert workbench.

17. Click the Expert Check icon (), rename the check, select the KWE language, and click OK. The Check Editor opens.

18. Enter the following script in the editor, then click **Apply** and **OK**.



Editor

```
udf1:UserFeature1 // UserFeature1 is the instance you indicated Step 10
Message("#", udf1.Axis) /*Axis is the name of the published parameter*/
```

19. Click the **Solve** icon (). A message box displays the distance to axis of each UDF.

The types generated when creating the UDF appear in the Object Browser (of the Expert Check Editor and of the Expert Rule Editor) in the package you selected in the **Manage Type** window.

Creating Rules and Checks in VB Script

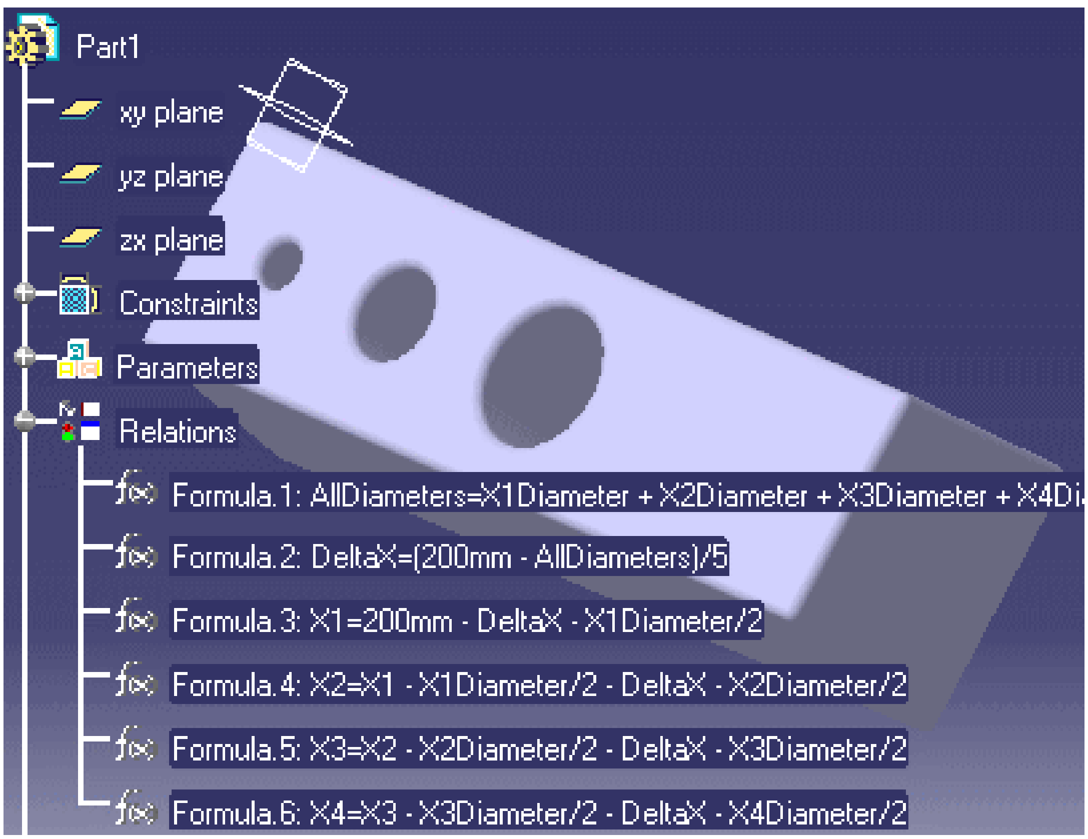
P2

The scenario below illustrates how to use a variable and specify a return value.

Instead of writing the body of expert rules and checks in the Knowledge Expert language, you can write it in VB script. There are some adjustments to be done:

- The variable declared in the (*for all*) field can be used as an object.
- The indicated type name in Knowledge Expert Language may change in VB script.
- To specify that a check is valid or not, set the Value attribute of the returnValue object (1 if the check is valid, 0 if the check is invalid).

1. Open the KwxUseCase1.CATPart document. The following screen is displayed.



2. Access the Knowledge Expert workbench, then click the icon. In the dialog box which is displayed, enter a check name and a comment. Select the Visual Basic language (KWE language is the default language), then click OK. The three tab editor is

displayed.

3. Select the Condition tab and enter the check below (comments starting with /* and ending with */ should not be used - if need be remove such comments).

```
(for all) H:Hole

Dim diam As Length
Set diam = H.Diameter

if (diam.Value >= 10.0 ) then
returnValue.Value = 1
else
returnValue.Value = 0
end if
```

- If no unit is indicated in VB script, the default unit will be mm; and m in Knowledge Expert Language.
- It is highly recommended not to add any comments (between '..') in VB script or a syntax error will be returned.

4. Click **OK** to add the check to the rule base, then click the icon to solve the rule base. In the specification tree, the check icon is red. This indicates that not all the part holes have a diameter greater than or equal to 10mm.

Workbench Description

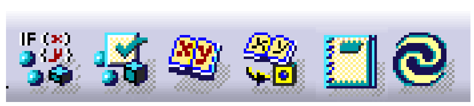
The Knowledge Expert Menu Bar

The menu bar which is available in the Knowledge Expert workbench is the standard one except the Insert command which provides you with the Expert Rule, Expert Check, Insert Rule Set and Insert Rule Base icons.

The Knowledge Expert Toolbar

The figure below shows the Knowledge Expert toolbar.

On the figure above, click an icon to display the documentation of the task associated with the icon.



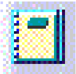
Here is a short description of each icon.

The ***Expert Rule*** icon provides access to the rule editor. Click this icon to create an expert rule, write its code, test its syntax and add it to your rule base.

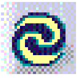
The ***Expert Check*** icon provides access to the check editor. Click this icon to create an expert check, write its code, tests its syntax and add it to your rule base.

The ***Rule Set*** icon is used to create a rule set. Click this icon to create a rule set below the rule base in the specification tree of your document.

The ***Insert Rule Base From Existing Document*** icon allows you to import a rule base from an external document. Click this icon to import in your current document the rule base (expert rules and expert checks) of an external document.



The ***Check Report*** icon is a means to generate a report. Clicking this icon is of interest when you have just solved a rule base with a certain number of checks applying to multiple features. The report gives you information on valid and invalid checks as well as extra information depending on the Rule Base Settings.



The ***Solve*** icon is to be used to solve a rule base. Click this icon to apply the rules and checks created in your rule base to your document.

Glossary

Many of the definitions included in this glossary are only pertinent within the knowledgeware context.

A E F K M O P R S

A

activity A property which defines whether a feature is applied to a document or not. The activity value is either `true` or `false`. It is indicated by an icon in the specification tree and can also be read in the document parameter list.

E

expert check A set of statements intended to give you a clue as to whether certain conditions are fulfilled or not. An expert check applies to the features of a given type. It does not modify the document it is applied to. An expert check is a feature. In the document specification tree, it is displayed as a relation that can be activated and deactivated. Like any feature, an expert check can be manipulated from its contextual menu.

expert rule A set of instructions, generally based on conditional statements, whereby the relationship between parameters is controlled. An expert check applies to the features of a given type. In the document specification tree, it is displayed as a relation that can be activated or deactivated. Like any feature, an expert rule can be manipulated from its contextual menu.

F

formula A relation specifying a constraint on a parameter. The formula relation is a one-line statement. Its left part is the parameter to be constrained, the right part is a relation taking as its variables other parameters. A formula is a feature. In the document specification tree, it is displayed as a relation that can be activated or deactivated. Like any feature, a formula can be manipulated from its contextual menu.

K

knowledgeware The set of software components dedicated to the creation and manipulation of knowledge-based information. Knowledge-based information consists of rules and other types of relations whereby designers can save their corporate know-how and reuse it later on to drive their design processes.

M

magnitude type parameter A parameter whose value is defined by a quantity expressed in specific units. Length, Angle, Time parameters are magnitude type parameters. Boolean, Real, String and Integer parameters are not magnitude type parameters.

O

object browser A form of user assistance that helps the user pick up objects such as features, feature attributes, operators and functions in a predefined list of objects.

P

parameter A feature defining a document property.

predicate The condition part in an expert rule. The other part of the expert rule describes the actions to be executed when the predicate is true.

R

relation A knowledgeware feature which, depending on certain conditions:

- sets parameter values
- displays a message
- or runs a macro.

Knowledgeware relations are formulas, checks, rules and design tables.

rule base The feature at the top of the expert rule/check hierarchy.

rule set A group of expert rules or checks

S

solve operation The operation which consists in applying all the rules/checks of a rule base to a document.

Index

A C D E F G H I K L M N O P Q R S T U V W

A

C

D

E

F

G

H

I

K

N

O

P

Q

R

S

T

U

V

W