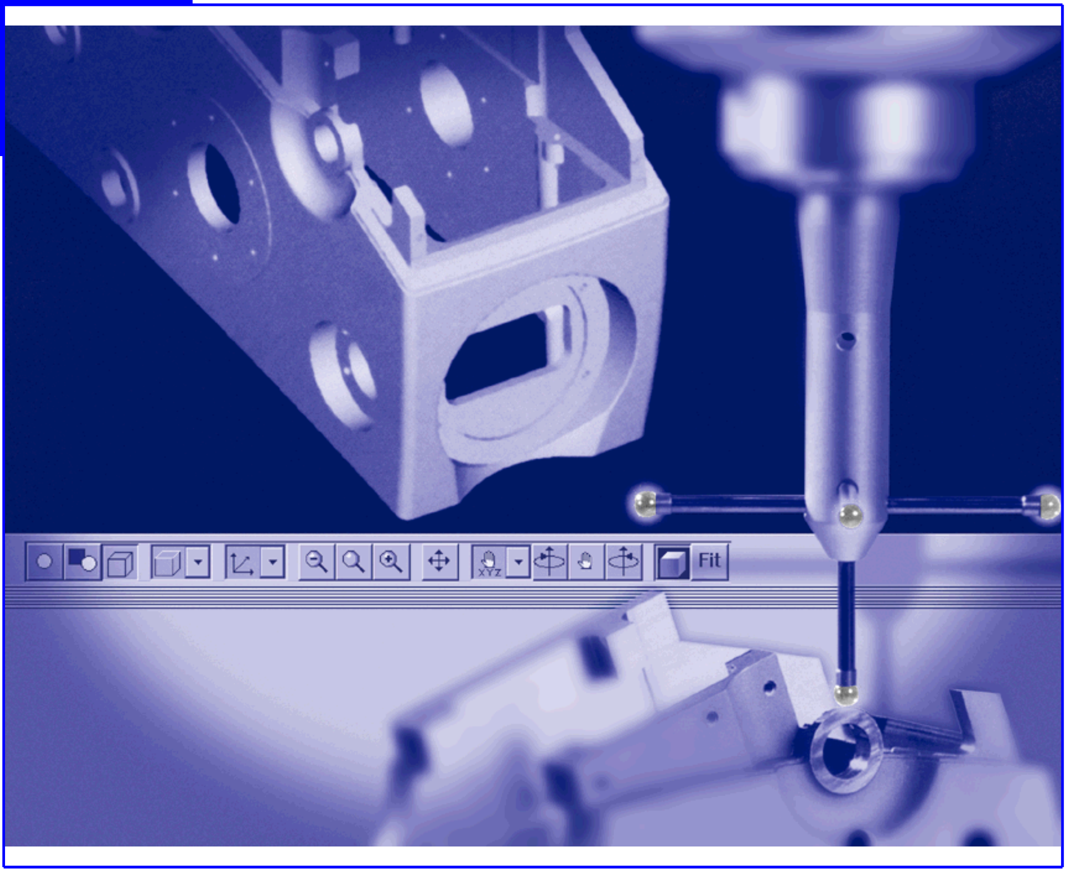
CALYPSO

Option 2
PCM technology



Operating Instructions

ZEISS

The design and delivered components of the CMM, its options, the program packages, and the relevant documentation are subject to change.

This manual must not be circulated or copied, or its contents utilized and disseminated, without our express written permission. Persons misusing this manual are subject to prosecution.

All rights reserved, especially in cases of granting a patent or registering a utility model.

This manual is subject to modification. All rights pertaining to changes in the CMM and its options reserved.

All product names are registered trademarks or trademarks of the corresponding proprietors.

Although utmost care has been taken in preparing the information given in this manual, we cannot assume any liability for its completeness and correctness, except in case of willful intent.

CALYPSO
Version 2018
Operating Instructions
2018-04-16
61212-2721502

Table of contents

Preface

Chapter 1 Parameter-coded measuring runs (option)

Preface

Information about these operating instructions

The CALYPSO program consists of a base module and additional options for special purposes. You can customize the scope of program to fit your requirements.

These operating instructions describe an option of CALYPSO and are based on the assumption that the user is familiar with the operating instructions for the base module of CALYPSO.

NOTE

The additional CALYPSO options are described in separate manuals.

Reference information about the windows and dialogs can be found in the dialog reference in the CALYPSO Online Help.

Simply Measure – And what you should know to do it right, A metrology primer

Carl Zeiss, Industrial Metrology Division,

Order no.: 612302-9002

Text conventions

The following text conventions are used in these instructions.

Example	Description
Features	Text element of the graphics screen display.
Comment	The **Comment** button on the screen.
<machine name>	Variable text or dummy for a name.
C:\windows\w.ini	The w.ini file in the windows directory on the C:\ drive.
For this section...	A passage containing important information.

Example	Description
➤ *Preface [➪ Preface-1]*	This is a cross reference. When viewing this manual on the screen, you will be guided to the indicated text passage by clicking the reference.
Plan → CNC-Start → Run	The **Run** command in the **CNC-Start** submenu of the **Plan** menu.
CTRL+A	Press the CTRL key and the letter A at the same time.

Icons

Three special symbols containing important information are used in this manual. The icons appear in the marginal column next to the respective text.

You will find a detailed explanation of the safety instructions under Configuration of safety instructions.

Configuration of safety instructions

Safety instructions indicate a personal health hazard. We distinguish three different levels: Danger, warning and caution. All three safety instructions are marked with the same warning symbol. The designation of the safety instruction is shown beside the symbol. The safety instructions used are described below.

Configuration of a safety instruction

A safety instruction may have the following components:

- Warning symbol and designation of the safety instruction (signal word): Danger, warning or caution.
- Source and cause of the danger
- Consequences for the user due to non-observance of the safety instruction
- Required measures to be taken by the user to avoid possible consequences
- A measure may cause an intermediate result.
- At the end of all measures, a final result may be caused.

Personal health hazard

⚠ DANGER

A »danger« indicates an imminent risk to life and limb.
Non-observance of this safety instruction when the described risk occurs causes death or serious injuries.
Example: Electric shock due to high electric voltage.

⚠ WARNING

A »warning« indicates a possible risk to life and limb.
Non-observance of this safety instruction when the described risk occurs may cause death or serious injuries.
Example: Risk of severe crushing of the body caused by heavy loads.

⚠ CAUTION

A »caution« indicates a personal health hazard.
Non-observance of this safety instruction when the described risk occurs may cause slight to moderate injuries.
Example: Risk of minor crushing of the limbs caused by small loads.

Risk of material damage

If there is no personal health hazard, but the CMM or components may get damaged, this is pointed out by the following notice.

This symbol refers to possible damage to the CMM.
Non-observance of this safety instruction when the event occurs may cause damage to the CMM or one of its components.
Example: Collision of the ram with a workpiece.

Chapter 1

Parameter-coded measuring runs (option)

This chapter contains:

Introduction to PCM

CALYPSO uses the PCM programming language for variable control of measuring runs. Of course you can use CALYPSO without PCM, but PCM offers you a powerful toolset for simplification and automation, along with part-variant programming and parameter-supported and interactive control of measuring runs.

What is PCM?

PCM is a programming language for parameterizing measuring runs. PCM is short for “parameter-coded measurement”.

PCM enables you to simplify and rationalize measuring runs in CALYPSO: You can program measurement plans in such a way that you can measure various workpiece variants, and you can set parameters to influence the automatic run of a measurement plan.

NOTE
On account of the complexity involved, programming in PCM is only recommended if you have basic knowledge of programming structures (e.g. Pascal or C).

PCM has functions for calculating values, for controlling the CMM, and for interactive input/output dialogs. You can vary the way in which a measurement plan runs by programming loops and conditions. In certain situations you can have CALYPSO generate special messages.

NOTE
A yellow input field in CALYPSO always contains a parameter or formula.

In CALYPSO, you use PCM when you enter a *Formula* in an input field for a feature. By so doing, you automatically define a variable and assign it parameters. The color of the input fields changes, in this case to yellow. By the same token, when you set a condition in CALYPSO or define a loop, you automatically create “PCM code”.

NOTE
You also have this possibility without the PCM Technology option.

If you use the PCM Technology option and you have familiarized yourself with the syntax, you can employ the PCM code to enter conditions and loops or numerous other commands directly or generate files for import into CALYPSO.

PCM in CALYPSO – first steps

Programming with PCM

Parameterizing measurement plans

The easiest and most reliable way of generating variable measurement plans is to begin by creating an ordinary measurement plan with fixed coordinates and values in CALYPSO in the usual way. You can then substitute variables for the specific coordinates that vary from workpiece to workpiece.

The various measuring runs you need derive from the formulas and parameters you enter for the variables.

Additional programming functions

Besides parameterization of measurement plans, PCM provides an extensive programming functionality:

- You can program dialogs that CALYPSO uses in the course of a CNC run to query the operator for quantities, so that the figures supplied in this way can subsequently be used in different ways as the run progresses.
- You can determine measured values and system parameters and reuse them in the measurement plan.
- In the case of conditions or loops, you are not bound to the forms specified in the dialogs – you can enter them directly with PCM.
- You can issue direct control and motion commands to the CMM.
- Finally, you can also enable external access to measured data via an interface.

You can use these functions in many different ways, individually and in combinations, so these instructions cannot cover all possible circumstances in detail – remember that PCM is a programming language, designed for you to use to the best possible effect.

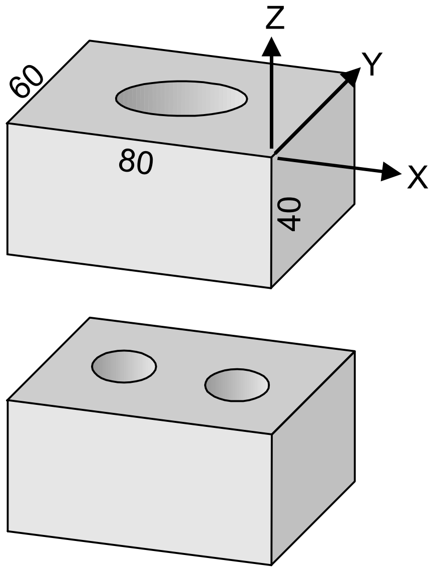
This chapter, therefore, describes the basics of the run structures of CALYPSO and the syntax of PCM, tells you how to enter parameters and formulas in CALYPSO, and contains a reference library of functions and commands.

The examples (see ➤ *Examples for PCM [➪ 1-40]*) demonstrate the versatility of PCM and show you how to utilize the performance of this programming language.

Variable measurement plans in PCM

Principle of parameterization

CALYPSO employs the principle of parameterization so that you can use a single measurement plan to measure workpieces in CALYPSO that are similar but not identical. This principle is based on the premise that identical values for both workpieces are defined in the measurement plan in CALYPSO as *fixed* values; those varying from workpiece to workpiece, on the other hand, are defined as *variables*.



In this way one and the same measurement plan can be used to measure and test different workpieces: for example, a plate with either one or two holes – even if the radii are different. Different geometries, differences in the number of certain features, the presence or absence of features - all these can be parameterized.

Parameters for variables in PCM

Note, however, that a measurement plan with variables cannot be run unless the variables are assigned specific values. These specific values, the parameters in other words, have to be known to CALYPSO: You define them in the measurement plan before it is run. CALYPSO then uses the coordinates or values entered for the variables in processing the measurement plan.

Example

The two workpieces in the illustration above are different, but “similar”: They have some identifying characteristics in common, while others are unique to each. The first workpiece in the illustration has a central hole with a radius of 20, while the hole in the second workpiece, while of the same depth, has a radius of 10 and is off-center. This workpiece, moreover, has a second hole.

The variables that could be used here are as follows: positions of the holes (hole1_center, hole2_center), their radii (hole1_radius, hole2_radius) and the number of holes (number_holes). These variables have to be assigned values for each workpiece, so that a measurement plan can be run:

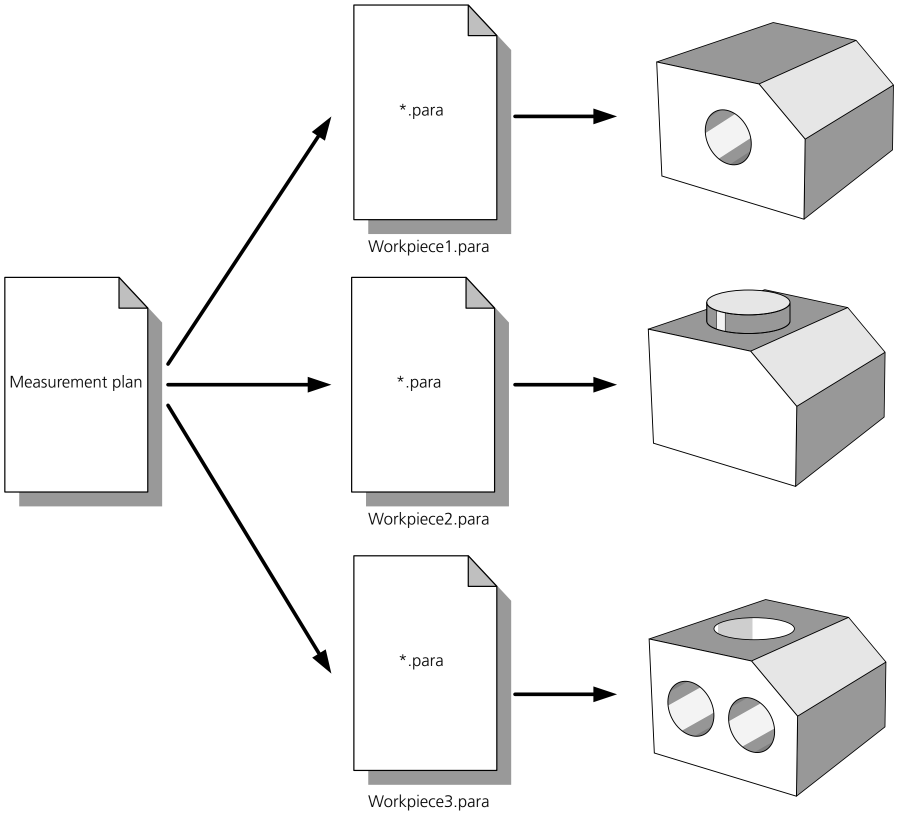
Variables	Parameters for workpiece No. 1	Parameters for workpiece No. 2
Number_holes	1	2
hole1_center	point(-40.30,40,0,0,1)	point(-20.30,40,0,0,1)
hole2_center	any	point(-60,30,40,0,0,1)
hole1_radius	20	10
hole2_radius	any	10

Set up in this way with only five different parameters, the measurement plan is able to measure two different workpieces.

Parameters in parameter files

The process is even more straightforward when you provide CALYPSO with the parameters of the current workpiece or the measurement plan variant you intend to use in the form of a prepared, reusable ASCII file. All you have to do when a new version of a workpiece is due for measurement is load a new ASCII file containing the required information.

Another advantage is that ASCII files can be edited independently of CALYPSO.



Parameterizing characteristics with PCM

Characteristics can sometimes be the same despite differences in features, as is the case, for example, with the diameter of a circle of the same size that only has to be measured at a different position on the workpiece. Other characteristics can vary as a function of the parameterized measurement plan.

This, in turn, means that if the value of the characteristic, and not just the measured values of a given feature, varies from workpiece to workpiece, you have to parameterize the characteristic as well.

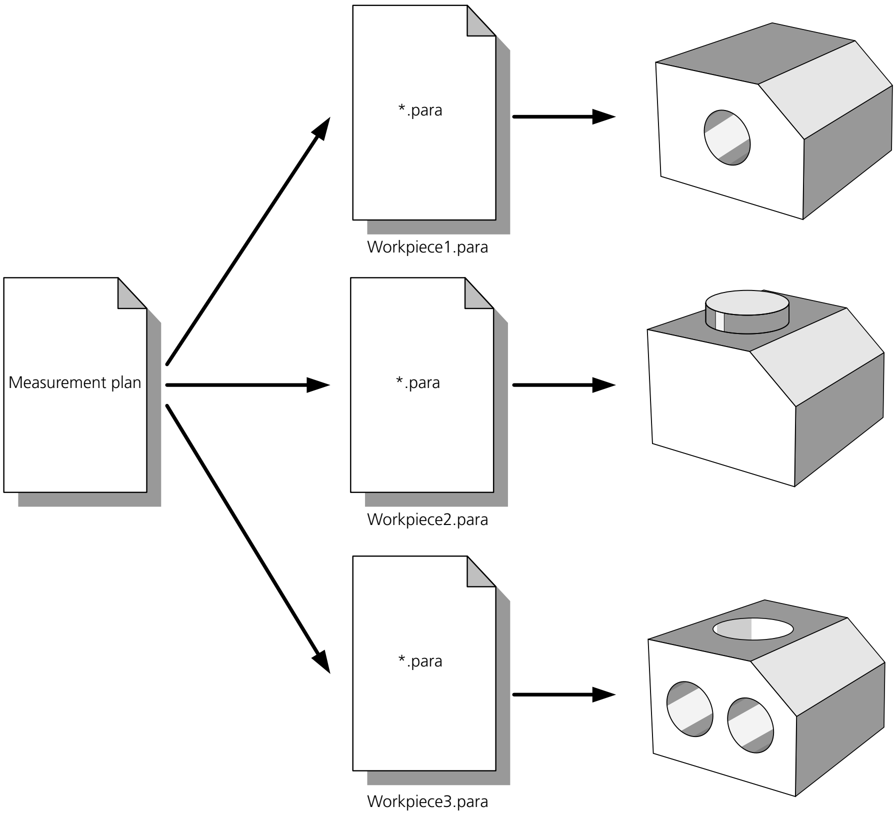
You do this by defining a variable for the characteristic in question and assigning it the corresponding value.

Parameter files for measurement plans

The advanced way of setting up variable measurement plans is to use *parameter files*. A parameter file contains the set of value assignments for the variables in a measurement plan.

When you parameterize fixed defining aspects of a workpiece, you are in a position to measure different workpieces with a single measurement plan: All you have to do is enter different parameters, either directly in CALYPSO or by specifying and loading a parameter file.

You can create different parameter files for a particular measurement plan and load whichever file you need for a given workpiece and the aspects you want to measure. You need a dedicated parameter file for each workpiece.



The parameter files can be created in two ways:

- You save the entered parameters of a measurement plan.
- You create the parameter file directly with an ASCII editor.

  To do so, you need experience with measurement plans and reliable knowledge of the PCM syntax. PCM uses a functional syntax similar to the programming languages Basic, C, Fortran etc. (see ➤ *PCM syntax [➪ 1-59]*).

  When you create a parameter file, it is important to remember that the file name extension has to be ".para", so that CALYPSO can recognize it as a parameter file.

Conditions and parameters for the CNC run with CALYPSO

CALYPSO starts running the measurement plan after CNC start. PCM commands and parameter assignments come into effect at a very wide variety of points.

- You can define “input and output parameters” for each feature and each characteristic, and for the measurement plan as a whole. This means that you can enter PCM commands when these elements are defined.

  The “input parameters” are processed before the element is run, the “output parameters” are processed after the run.
- You also have the opportunity of setting “start parameters” once or loading a file before the measurement plan is run.
- You can set a condition and a loop for each characteristic.
- You can enter “input and output parameters” for each condition and each loop.

Utilizing these CALYPSO functions in this way, therefore, you can construct complex, nested structures.

Wherever you set input parameters, output parameters or start parameters, moreover, you can program conditions and queries directly with PCM.

In addition, you can have external PCM files run in certain situations of the measurement plan run without any intervention in the measurement plan.

Sequence of PCM parameter processing

The sequence adopted by CALYPSO to process the data is as follows:

1. The input parameters are evaluated before the measurement plan is run; the output parameters are evaluated when the run completes.
2. CALYPSO checks each characteristic in succession, in the order in which it appears in the list of characteristics. The condition and the loop are processed first (depending on the order in which they were entered) followed by the input parameters of the characteristic. Once the characteristic has been determined, its output parameters are evaluated.
3. Before a feature defined for a characteristic is measured, its input parameters are evaluated. The feature's output parameters are evaluated when measurement completes.

4. If input or output parameters are set for a condition or loop, the input parameters are evaluated before the condition is checked or the loop is run. The output parameters are evaluated after the element referenced by the condition or loop.

   If the condition results in a stop, the output parameters are not evaluated.

You might find it useful to visualize the input and output parameters as “parentheses” bracketing the objects in question.

NOTE

In addition, CALYPSO checks prior to, during or after a measurement plan run whether a certain external file with a defined name exists in the current measurement plan directory or in the general directory for measurement plans. This file is then executed.

Variables, parameters and formulas in CALYPSO

What are variables and parameters in PCM?

If you want to run variable measurement plans for different workpieces in CALYPSO, you have to use variables instead of absolute values to describe the workpiece.

What are variables?

Variables, by definition, are quantities that may vary. By means of PCM, all defining aspects required in descriptions of workpieces or runs in measurement plans can be characterized as variables in CALYPSO.

The variables are stored “locally” together with the element (feature, characteristic, condition, loop) for which they were defined.

The assigned values, the parameters in other words, are stored centrally for the entire measurement plan.

Types of variables

PCM recognizes six types of variables: Numbers, points, vectors, character strings, logical values, and lists. Each variable has a name comprised of alphanumeric characters (without spaces; first character is not a number), and different variables have different names.

Examples: radius_1, set-down point, message_354

Arrays

Groups of variables are called arrays. The only difference in the names of variables in an array is the array index. Example:

radius[1], radius[2], radius[3], radius[4]

In this case, “radius” is an array with four variables.

What are parameters?

Parameters are specific values for variables. When variables are used in a measurement plan instead of fixed numbers or texts, you can vary the measurement plan by assigning certain fixed values (parameters) to the individual variables.

You assign values either by entering them directly or by means of statements in a file. Use the following syntax:

```
<Variable name>=value
```

You can enter the value using one of the following formats:

Variable type	Format for values (example)
number	-3.141529
vector	vector(10,12,0)
point	point(-10,12.5,0,0,0,1)
string	"Circle"
Boolean	True
list	list(3.1416,3.4242,-5.2323)

NOTE

The last three components of a variable of the “point” type represent the coordinates of a normalized vector. This means that the root of the sum of the three squares (= the length of the vector) must always be “1”.

It follows that the PCM commands listed below are value assignments:

- P1 = 10
- vector_1 = vector(10,20,30)
- corner_1 = point(10.1,20.5,30.02,0,0,1)
- text_1= "This is a text in the text_1 variable"
- In the Multiple Printouts mode = true
- LIST = list(3.1416,2.71828,0,0,0)

Value assignments and variable definitions

A value assignment also defines the variable and makes it known to CALYPSO. There are no explicit variable definitions (constructors) in PCM.

If, in the measurement plan run, a variable that has not yet been defined is to be accessed, a warning message appears.

Variables in the arrays can be defined either individually “one by one” or in a loop instruction. Example:

- array[1] = point(-10,12.5,0,0,0,1)

  array[2] = point(-12,12.5,0,0,0,1)

  array[3] = point(-14,12.5,0,0,0,1)

  array[4] = point(-16,12.5,0,0,0,1)

Here, "array" has been defined as an array consisting of four variables of the point type.

Formulas and functions in PCM

As well as using individual variables in measurement plans, you also have the option of using functions of variables and formulas with variables.

Return values

The return value is an important function of a variable: the individual parts of a parameter have to be “addressed” so that you can work with them.

The return values of variables of the “point” type are defined as follows:

Function	Return value
variable_name.x	x value of the “point” variable
variable_name.y	y value of the “point” variable
variable_name.z	z value of the “point” variable
variable_name.nx	x value of the normal vector of the “point” variable
variable_name.ny	y value of the normal vector of the “point” variable
variable_name.nz	z value of the normal vector of the “point” variable

You can access variables of the vector type as follows:

Function	Return value
variable_name.x	x value of the “vector” variable
variable_name.y	y value of the “vector” variable
variable_name.z	z value of the “vector” variable

Arithmetic operators and functions

The most important operators and functions with variables are:

- Arithmetic operators:

  +,-,*,/

- Comparative operators (used in conditions and loops):

  <,>,==, <>, <=, >=

- Mathematical and trigonometric functions:

  sqrt, squared, abs, exp, int, ln, log, mod, round, sign,

  sin, cos, tan, arcsin, arccos, arctan, arctan2

- Functions for determining values (query):

  getNominal, getActual

– Character string functions:

  +, asc, chr, format, inStr, len, mid, strElement, val

There are also a number of functions for input and output, for run control and for working with files, plus measurement-specific and CMM-specific commands, system commands, CMM movement commands as well as conditions and loops (see ➤ *PCM quick reference [➪ 1-58]*).

Programming with PCM

Basics about programming

Programming conditions and loops with CALYPSO

You can vary the way in which the measurement plan runs by setting conditions and loops.

- A characteristic or feature with a condition is not executed unless the condition is satisfied. The characteristic is either calculated or not calculated depending on the truth value returned for the condition, which means that the measurement plan run either proceeds or is interrupted.
- A loop around a characteristic or a feature results in the characteristic or feature being run repeatedly, possibly with changed variables for each run.

You trigger these functions by selecting `Condition` or `Loop` in the context menu and entering the settings for the condition or loop, as applicable (see Setting conditions and Placing loops in the operating instructions for the base module). Furthermore, you can define conditions and loops in the function and parameter list. (See PCM functions: Conditions/ Loops)

Programming conditions and loops with PCM

Over and above the options at your disposal in CALYPSO, you can use PCM to insert conditions and loops explicitly in parameter files or in the input and output parameters of features or characteristics.

In this way you can utilize conditions and loops much more comprehensively – you are in control of the measuring run. During the CNC run, the truth values of the conditions are determined, the loop index is incremented automatically and the definitions, the value allocations and the commands are processed accordingly.

The following control elements are available for both the input and output parameters:

- the ➤ *Simple condition [➪ 1-14]* (if - endif)
- the ➤ *Condition with alternative [➪ 1-14]* (if - else)
- the ➤ *Condition with several alternatives [➪ 1-15]* (selectCase - endSelect)
- the ➤ *Specific loops [➪ 1-16]* (for - next)
- the ➤ *Conditioned loop [➪ 1-17]* (repeat - until)

Simple condition

The syntax of the simple condition is as follows:

```
if CONDITION then
   DEFINITION
endif
```

Meaning:

- If CONDITION is satisfied, DEFINITION is processed.
- If CONDITION is not satisfied, DEFINITION is not processed.

The DEFINITION can be formulas, value assignments, functions or other conditions or loops of your choice, always with line breaks as separators.

Condition with alternative

The syntax of the condition with alternative is as follows:

```
if CONDITION then
      DEFINITION1
   else
      DEFINITION2
endif
```

Meaning:

- If CONDITION is satisfied, DEFINITION1 is processed.
- If CONDITION is not satisfied, DEFINITION2 is processed.

DEFINITION1 and DEFINITION2 can be formulas, value assignments, functions or other conditions or loops of your choice, always with line breaks as separators.

Example

In the example below, the “result” variable is assigned the diameter of circle circle_1. A message corresponding to this quantity is then output on the screen:

```
result=getActual("circle_1").diameter
if result>10 then
      message(“diameter circle_1 is greater than 10")
   else
   if result<10
then
      message("diameter circle_1 is smaller than 10")
   endif
endif
```

Condition with several alternatives

The condition with several alternatives makes it possible to program a branch. The syntax is:

```
selectCase TEST PRINTOUT
   case VALUE LIST 1
     STATEMENTS 1
   [case VALUE LIST 2
      [STATEMENTS 2]]
   ...
   [caseElse
     [ELSE STATEMENTS]]
endSelect
```

Meaning:

- TEST PRINTOUT:

  Required. Any numeric expression, character string or PCM parameter.

- VALUE LIST:

  Must be specified after `case`.

  A list separated by commas that contains one or more of the following forms:

  - Expression:
    e.g. `12`, "`All`"
  - Expression1 `to` Expression2:
    e.g. `1 to 15`, "`Part1" to "Part5`"
    The operator `to` defines a range of values. The smaller value must be on the left. With a character string, the lexicographical order applies.
  - `is` comparative operator expression:
    e.g. `is > 8`

- STATEMENTS:

  Optional. One or more statements that are carried out if TEST PRINTOUT corresponds to any element in the corresponding VALUE LIST. The condition with several alternatives is browsed from top to bottom to find a match between TEST PRINTOUT and VALUE LIST. The STATEMENT of the first VALUE LIST that corresponds to TEST PRINTOUT is executed. At the end of `endSelect`, the operation continues.

– ELSE STATEMENTS:

  Optional. One or more STATEMENTS that are carried out if no match between TEST PRINTOUT and a VALUE LIST is found in the `case` sections.

NOTE

Conditions with several alternatives can be nested.

For example, a condition with several alternatives can look as follows:

```
selectCase number
      case is <= 0
            Message = "Number is too small"
      case 1, 2, 3
            Message = "Number is smaller than 4"
      case 4 to 7
            Message = "Number is between 4 and 7"
      case is >8 AND number < 11
            Message = "Number is 9 or 10"
      caseElse
            Message = "Number is outside the range"
endSelect
```

Defined loop

In the case of the defined loop, the number of loops is defined unequivocally. The syntax of the defined loop is as follows:

```
for index=start to end [step]
   DEFINITION
next index
```

The following must be inserted:

– for `index` – the name of the loop variables (random),

– for `start`, `end` and `step` – whole numbers,

– for `DEFINITION` – random formulas, value assignments, functions or other conditions or loops of your choice, always with line breaks as separators.

Meaning: The functions or commands under DEFINITION are processed (end - start + 1)/step times, whereby “step” is set to 1 if no other specification is made. At the same time, the specific value for index is always entered in DEFINITION: at first start, then start+step, start+2*step etc. up to end. With next i, the loop index is incremented by step each time.

Example

```
for i=1 to 4
   message(i,". step: ",step[i])
next i
```

Conditional loop

In the case of the conditional loop, the number of runs depends on the fulfillment of a condition, whereby the condition is only tested after the first run. Therefore a conditional loop must run at least once. The syntax of the conditional loop is as follows:

```
repeat DEFINITION until CONDITION
```

Meaning:

– DEFINITION is processed. Then CONDITION is tested.

– If CONDITION is fulfilled, the loop is ended.

– If CONDITION is not satisfied, DEFINITION is processed again and CONDITION is tested again.

The CONDITION can be set up as a logical combination of several subconditions.

The DEFINITION can be formulas, value assignments, functions or other conditions or loops of your choice, always with line breaks as separators.

NOTE

Please note that infinite repetitions are possible with the conditional loop if the condition is never fulfilled.

Run control via parameters in PCM

Run control via parameters

Control in CALYPSO

In CALYPSO, you have the following possibility to control the run without using the PCM Technology option:

- You can enter formulas directly.
- You can conveniently transfer functions and parameters from the function and parameter list into the **Formula** dialog box.
- You can test the value of a formula.
- You can parameterize the space axis.
- You can set a condition for each individual characteristic.
- You can place a loop around the entire measurement plan and around each individual characteristic to force repeated runs.

These functions are described under Formula input and run control in the operating instructions for the base module.

Control in PCM

The PCM Technology option offers the following additional functions:

- You can enter parameters for the variables of a measurement plan.
- You can use PCM variables.
- You can enter input and output parameters for features, characteristics, conditions and loops.
- You can use the point generator to read in the points of a curve or define them with a mathematical expression.
- You can enter global measurement plan parameters or load them from external parameter files.
- You can define PCM test settings.
- You can start external programs and execute PCM files.

Entering parameters for measurement plans

Options of parameter inputs

You enter parameters for a measurement plan in the **Settings** window. In so doing you assign formulas or values to existing or newly defined variables.

You can enter the parameters directly in the window or you can load them from a parameter file.

You can create a parameter file by saving the parameters of a measurement plan or by saving a PCM code that has been created in the PCM editor. Additionally, you can explicitly generate a parameter file with an ASCII editor.

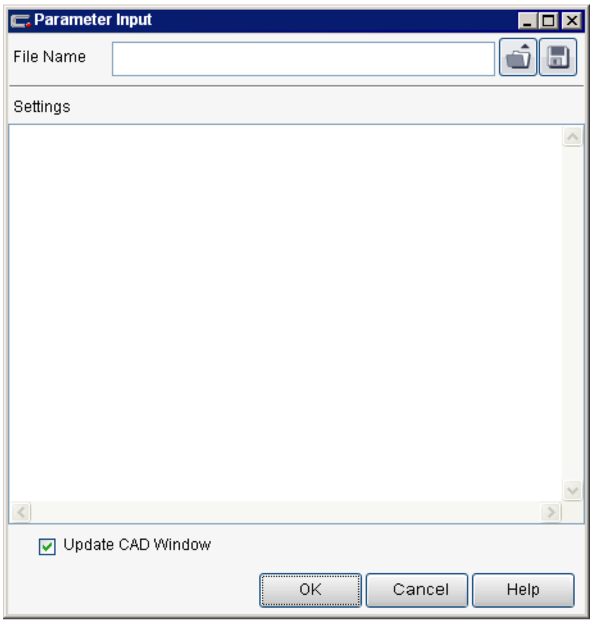
NOTE

In PCM and in the formula, the decimal separator is always a period ("."; example: "3.5 + 4.8", and *not*: „3.5 + 4.8").

The comma is used as the separator within value assignments (example: Location = point(3.5, 4.5, 1.5, 0, 0, 1)).

How to enter parameters for a measurement plan directly

1 Select **Plan** → **Advanced** → **Parameter**.
The **Parameter Input** dialog box opens.



☑ Update CAD Window

2 Click in the (white) input field and start defining the parameters:

- Start with the variable name, followed by "=".
- Proceed with the definition. Please observe the syntax described in ➤ *Formulas and functions [⇨ 1-12]* as well as the parameter syntax (see ➤ *Example for PCM: ASCII parameter file [⇨ 1-53]*).

3 Click **OK**.

The window will be closed. The parameter is now included in the list of all available parameters (this dialog box is fully described in Formula in the CALYPSO dialog reference in the Online Help).

How to save the measurement plan parameters in a file

Once parameters have been defined in a measurement plan, you can save them in a file. In this way, they are readily available for subsequent use in other measurement plans.

1 Select **Plan** → **Advanced** → **Parameter**.
The **Parameter Input** dialog box opens.

2 If necessary, edit the parameters, value assignments and formulas shown.

3 Click the diskette icon.
The **Save: Select Parameter File** dialog box appears on the screen.

4 Enter the name of the file (do not omit the ".para" extension), select a directory and click **Save**.

The variables and parameters you selected are saved in the file.

How to load measurement plan parameters saved beforehand

You can integrate parameters saved in a parameter file ("*.para*" file name) into your current measurement plan.

Opening a *.para file

1 Check that the **Parameter Input** dialog box is open (**Plan** → **Advanced** → **Parameter**).

2 Click the **Open** icon.
The **Open: Select Parameter File** dialog box appears on the screen.

3 Select the appropriate file and click **Open**.

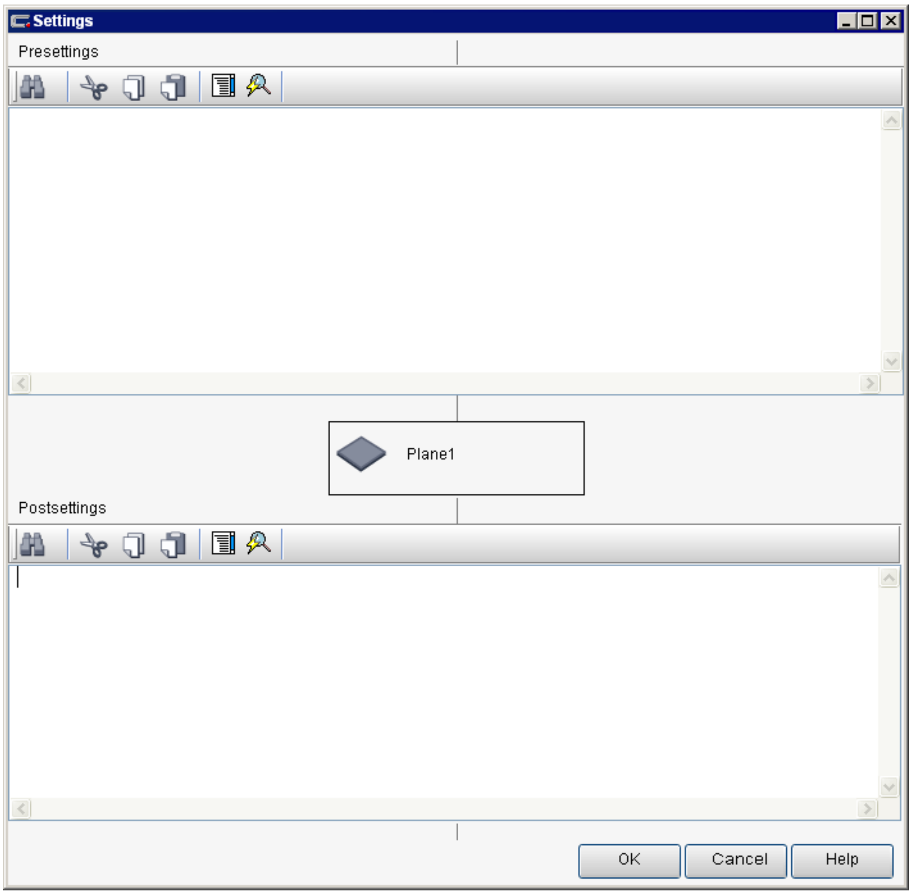
The parameters in the file are listed in the input window and are available for your current measurement plan.

Input and output parameters in PCM

Settings (Parameter) window

Input and output parameters can be set for each characteristic and feature, and for each condition and loop. The settings in the input and output parameters can be used to set temporary variables, print values in the default printout, and trigger functions.

Input and output parameters are defined in the Settings (Parameter window.



You can define input and output parameters for the following objects in CALYPSO:

- an entire measurement plan,
- a macro,
- a characteristic,
- a feature,
- a condition,
- a loop.

The entire PCM syntax is at your disposal for defining input and output parameters.

Comments

You can also enter comments. They will be displayed in color.

All characters of a line that start with a double backslash (//) will be interpreted as a comment.

For more information on the Settings (Parameter) window, please refer to the CALYPSO dialog reference in the Online Help.

Evaluation of input and output parameters

The commands and value assignments you enter as input parameters are evaluated before the object in question is executed or evaluated.

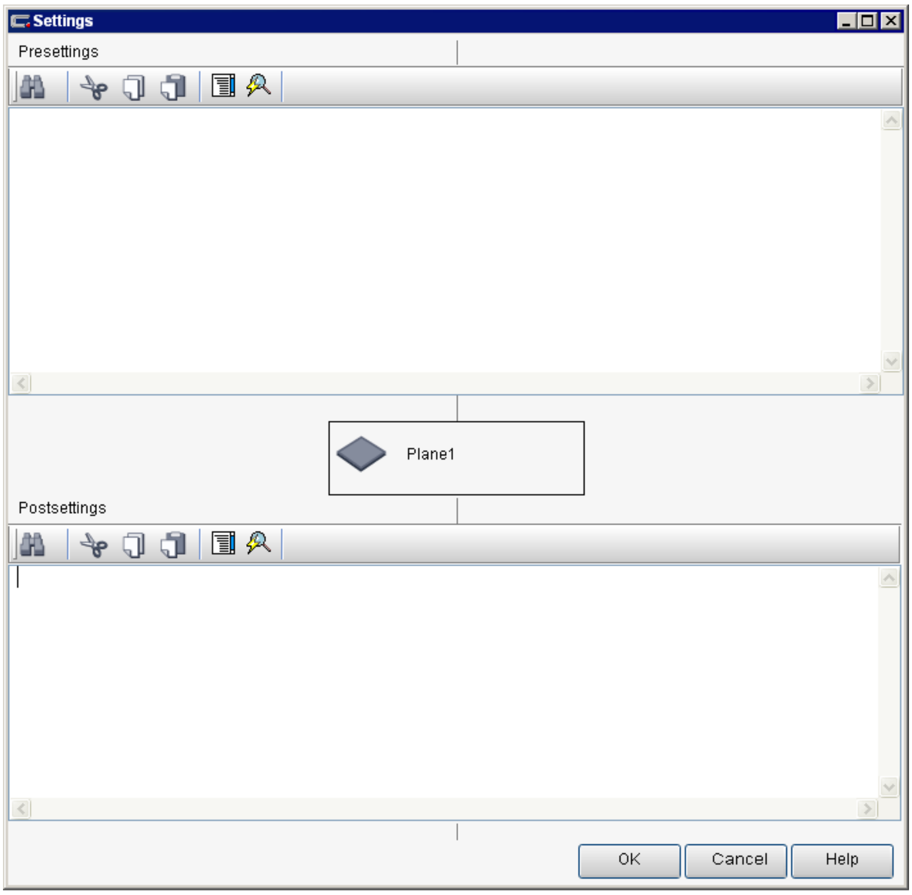
The commands and value assignments you enter as output parameters are evaluated after the object in question has been executed or evaluated.

An object, in turn, can consist of several other objects or be superior to these lower-order objects, so this means that under certain circumstances the input parameters and then the output parameters of the lower-order objects might be evaluated first.

If the object in question is a condition and the condition leads to omission of the characteristic or cancelation of the CNC run, the output parameters of the condition are *not* evaluated.

Entering input and output parameters

In CALYPSO, you can set input and output parameters for each feature, characteristic, condition or loop and for the entire measurement plan. To do so, use the **Settings** window.



1 To open the settings for the entire measurement plan:

- Open the list of characteristics and make sure that no characteristics are highlighted.
- Right-click and select **Parameter** from the context menu.

  Or
- P Open the **Open parameters of selected feature** dialog box.

2 To open the settings for a specific characteristic or feature:

- Highlight the characteristic or feature in the list of characteristics or features.
- Right-click and select **Parameter** from the context menu.

  Or
- P Open the **Open parameters of selected feature** dialog box.

3 To open the window for a condition:

- In the **Condition** window, click **Settings**.

4 To open the window for a loop:

- In the **Loop** window, click **Settings**.

The Settings (Parameter) window appears on the screen.

5 Enter the input parameters in the box at the top and the output parameters in the box at the bottom.

6 Click **OK**.

The window closes and your settings are accepted.

Entering input and output parameters using a list

You can also enter the parameters for presettings and postsettings in the **Settings** dialog box using a list. To do this, use the `inquireParameterList(...)` command.

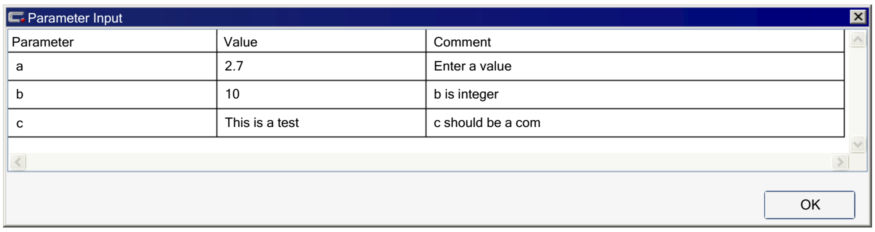
Prerequisite: A line containing the following command is contained in the respective settings:

```
inquireParameterList("parameter_name1","comment1",...)
```

All variables that are to be entered using the list must be listed in the brackets.

1 Select the line with the `inquireParameterList(...)` command.

2 In the context menu, select **Calculate**.
The **Parameter Input** dialog box appears on the screen with a list of the parameters that are contained in the brackets of the `inquireParameterList` command.



Parameter	Value	Comment
a	2.7	Enter a value
b	10	b is integer
c	This is a test	c should be a com

3 Enter the currently required values for the parameters or overwrite the default values.
Note: During the measuring run, you can overwrite only the entries in the **Value** column.

4 Press **OK** to confirm.

The dialog box is closed and the parameter values are accepted.

Parameterizing space axis

You can parameterize the space axis that is to be specified in various features. Select in the context menu the **Formula** input field and enter the desired space axis in the form of a formula. The result given by the formula must be a number.

During run time, the result is calculated by CALYPSO and interpreted as axis according to the following table:

Result	Space Axis
-3	-Z
-2	-Y
-1	-X
1	X
2	Y
3	Z
all other values	Z

The result of the formula defines the direction of the feature geometry. The numerical values entered for the projected angles will not change. After input of a formula, the text field will be highlighted in yellow.

The formula is evaluated in the CNC run during run time, the result of the formula is then used as space axis of the feature. The normal vector geometry thus changes during run time.

NOTE

It is also possible to change the space axis, i.e. the normal vector geometry during run time by a parameterized normal vector of the feature. The parameterized normal vector has a higher priority than the parameterized space axis.

Properly terminating the measurement plan

PCM enables you to properly terminate the measurement plan as a function of the situation. To do this, use the *endInspection()* PCM function.

As soon as this function is called in the CNC run, the current feature or characteristic will be processed completely, all intended records will be generated and output and the measurement plan run will be concluded.

NOTE

While the *cncBreak()* command is being processed, the CMM just stops. Printouts will not be generated.

Testing a PCM expression

You can test a PCM expression in order to ascertain whether the results are in line with your expectations or whether the PCM expression requires improvement.

1 Make sure that the **Settings** window is open with the PCM expression.

❗ Note that movement commands are executed by the CMM and that collisions could result. Move the CMM slowly and stop in good time if necessary.

2 Highlight the PCM expression in question and right-click.

3 Select **Compute** from the context menu.
The highlighted expression is computed or executed on the basis of the currently valid values of the variables.

If an error is encountered or parameters for variables have not been defined, messages to this effect are issued.

Run control with external PCM files

Run control with external PCM files

PCM allows you to start external programs from within the measurement plan as well as execute PCM files, without storing the call in the measurement plan.

This function makes it possible to change certain settings within one or all measurement plans by means of PCM functions without changing the measurement plans.

In addition, you can execute external batch files (see Run control with external batch files in the operating instructions for the base module). You can use two different file types:

File type	Syntax	File name
Batch file	according to Windows	*.bat
PCM file	PCM syntax	*_pcm.txt

Conditions for the execution

The external file is executed automatically in a certain CNC run situation if it has a certain name and is saved in a certain directory. You do not need to call up the file in the measurement plan.

- If the file is stored in the general directory for measurement plans (<*user directory*>*workarea**inspections*) or actual values, the file is executed each time a measurement plan is run.
- If the file is stored in the general measurement plan directory (<*user directory*>*workarea**inspections*\<*measurement plan name*>) or in the actual values directory, the file is executed when the corresponding measurement plan is run.

Order of the run

If several external PCM files and external batch files with the same name exist, the files are processed in the following order:

- PCM file in the general directory for measurement plans
- PCM file in the current measurement plan directory
- Batch file in the current measurement plan directory
- Batch file in the current directory for measurement plans
- PCM file in the current actual values directory (if this deviates from the general directory for measurement plans)
- PCM file in the current actual values directory (if this deviates from the measurement plan directory)
- Batch file in the current actual values directory (if this deviates from the measurement plan directory)
- Batch file in the current actual values directory (if this deviates from the general directory for measurement plans)

NOTE

By saving external files with the corresponding names in the general directory for measurement plans, you change the run of all measurement plans on the system without having changed the measurement plans.

Limiting the validity of external PCM files

If an external PCM file in the general directory for measurement plans should not be executed for all measurement plans, you can control this via a parameter query after the measurement plan name. Example:

```
if getRecordHead("planid") = "testplan"

<Series of PCM commands>

endif
```

You can also request the operator, creation data, subject number, etc. and use them as criteria.

Reference: Names of external PCM files

PCM allows you to start external programs from within the measurement plan as well as execute PCM files, without storing the call in the measurement plan.

This function makes it possible to change certain settings within one or all measurement plans by means of PCM functions without changing the measurement plans.

The name of the external files defines where in the measurement plan run the file is executed. You can use the following file names:

Call	Batch file	PCM file
Prior to the start dialog	Inspection_pre_start_dialog.bat	inspection_pre_start_dialog_pcm.txt
After loading the measurement plan	inspection_post_load.bat	inspection_post_load_pcm.txt
At the beginning of the run	inspection_start.bat	inspection_start_pcm.txt
After the end of the measurement; during the run after features prior to the computation of characteristics	measurement_end.bat	measurement_end_pcm.txt
Prior to the execution of the input parameters of each feature	–	plugin_preFeature_pcm.txt
Prior to the execution of the output parameters of each feature	–	plugin_postFeature_pcm.txt
Prior to the execution of the input parameters of each characteristic	–	plugin_preCharacteristic_pcm.txt
Prior to the execution of the output parameters of each characteristic	–	plugin_postCharacteristic_pcm.txt
After the end of the computation, but prior to the report/file output (not during the run with a selection of features)	calculation_end.bat	calculation_end_pcm.txt

Call	Batch file	PCM file
After the end of the run and the report output	inspection_end.bat	inspection_end_pcm.txt
After completion of all (also externally) created reports and files, if they are started by CALYPSO (e.g. PDF, DFD/DFX) (not during the run with a selection of features)	report_end.bat	report_end_pcm.txt
Prior to saving the measurement plan	inspection_pre_save.bat	inspection_pre_save_pcm.txt

Working with PCM

Overview: Working with PCM

The PCM programming language provides you with a wide choice of functions and commands that you can use to automate many of the routine procedures in CALYPSO.

You can edit the PCM code in the PCM editor (see ➤ *Working with the PCM editor [➪ 1-30]*)

You can use the search function to search for certain PCM texts or features with formulas, parameters or directory names (see ➤ *Searching in the measurement plan [➪ 1-32]*).

Additional debug functions are useful for locating bugs in programs (see ➤ *Carrying out PCM test settings [➪ 1-38]*).

You can also have the calculated formulas, presettings and postsettings output in the default printout. This is an additional option to verify your programming (see ➤ *Showing formulas in the default printout [➪ 1-39]*).

Working with the PCM editor

The CALYPSO PCM editor allows you to create the PCM code independently of features and characteristics. You do not have to create the PCM code in a separate program.

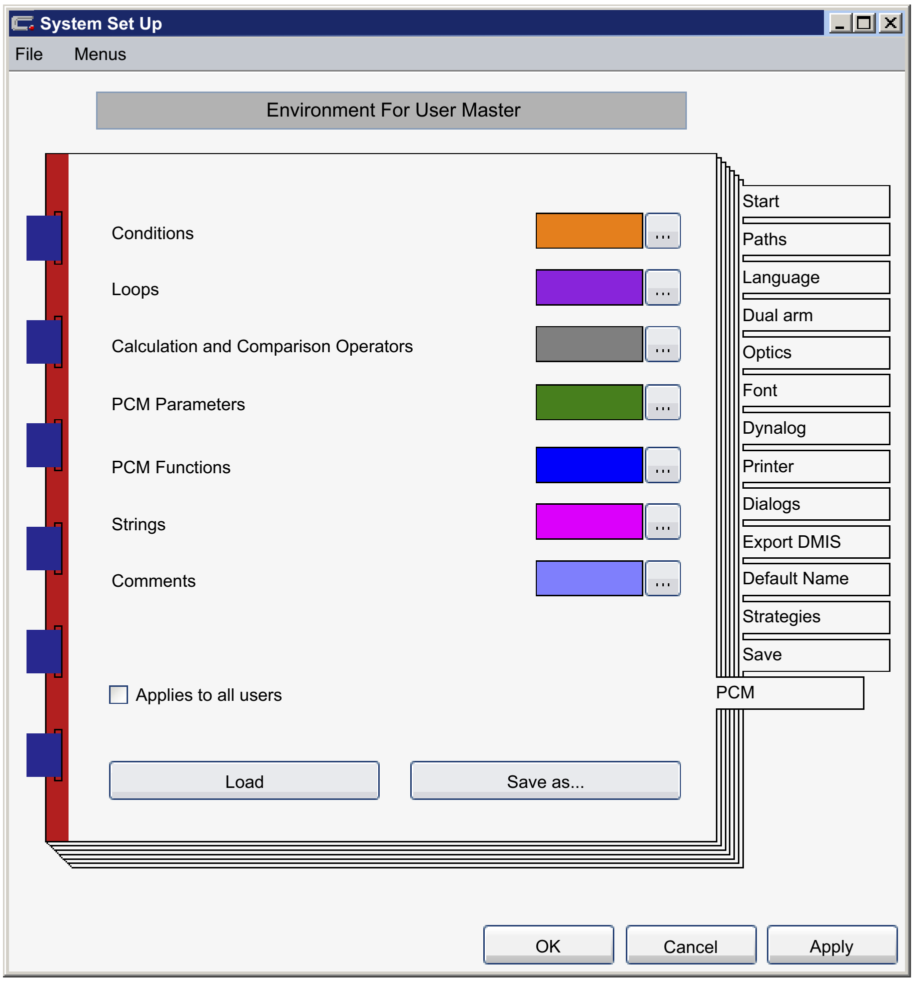
You simply enter the PCM code in the PCM editor of CALYPSO where you can edit it afterwards. The PCM editor offers several functions and settings facilitating the entry of PCM expressions.

Presettings for the PCM editor

You can highlight individual PCM expressions in color in the PCM editor. For example, you can choose to display all PCM parameters in green color and all PCM functions in blue color. The PCM code will then be clearer.

These color settings also have effect on the entries in the presettings and postsettings.

To preset the color highlight in the PCM code, select **Extras** → **Settings** → **Environment**. Select the **PCM** bookmark in the notebook.



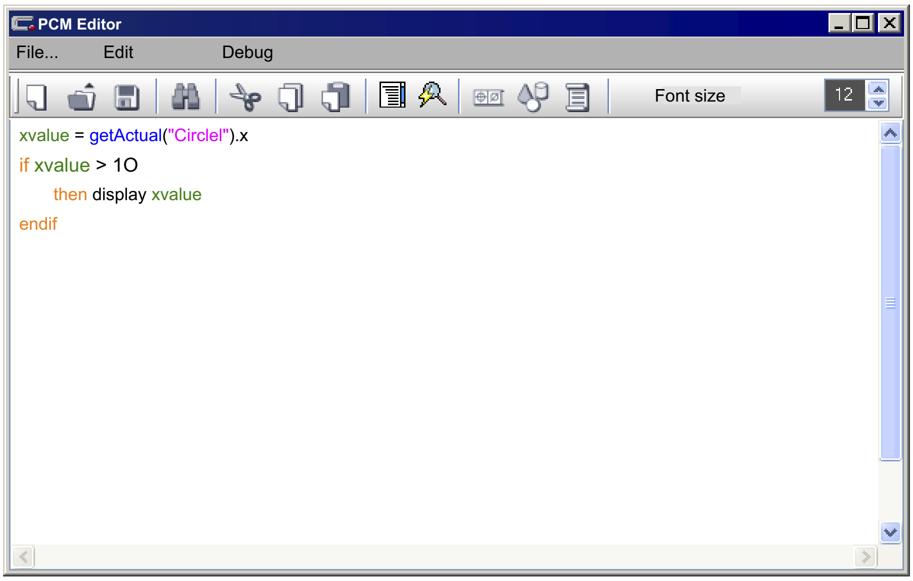
Conditions 🟧
Loops 🟪
Calculation and Comparison Operators ⬛
PCM Parameters 🟩
PCM Functions 🟦
Strings 🟪
Comments 🟦

☐ Applies to all users

You will find more information on the color highlight of PCM expressions under System Set Up (Environment) - PCM in the CALYPSO dialog reference of the Online Help.

Editing the PCM code in the PCM editor

To open the PCM editor, select **Plan** → **Advanced** → **PCM Editor**.



```
xvalue = getActual("Circle1").x
if xvalue > 10
    then display xvalue
endif
```

Entering the PCM code

You have two options of entering PCM functions and commands:

– Enter the code manually in the text field of the PCM editor.

  You can find the PCM expressions under ➤ *Overview: PCM quick reference [➪ 1-58]*.

– Insert codes from the **function and parameter list** and edit them afterwards.

  You can use the predefined PCM expression from the **function and parameter list**. Open the **Function and parameter list** using **Edit → Function**.

Reusing the PCM code

You can save the PCM code in the *.txt* format.

The saved file with the PCM code can then be loaded by you or other CALYPSO users in the PCM editor.

The PCM code can be copied from the PCM editor to the clipboard and pasted to the input and output parameters for features and characteristics.

Searching in the measurement plan

CALYPSO offers several search functions applicable to the entire measurement plan.

– You can look for certain character strings in PCM commands.

– You can look for features with formulas.

– You can look for features with parameters (presettings and postsettings).

– You can look for features and files with directory names.

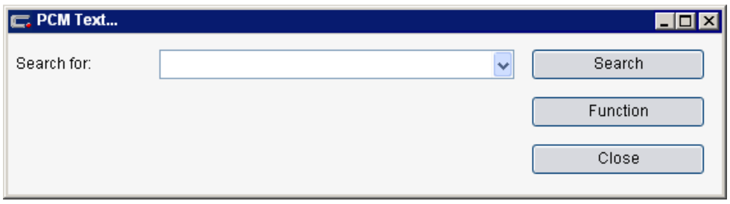
CALYPSO displays the search results and enables further operations with the search results:

- Double-click a displayed feature to highlight it in the measurement plan.
- You can search the search results for certain texts.
- You can copy the search results into a file for further use.

Searching for a PCM text

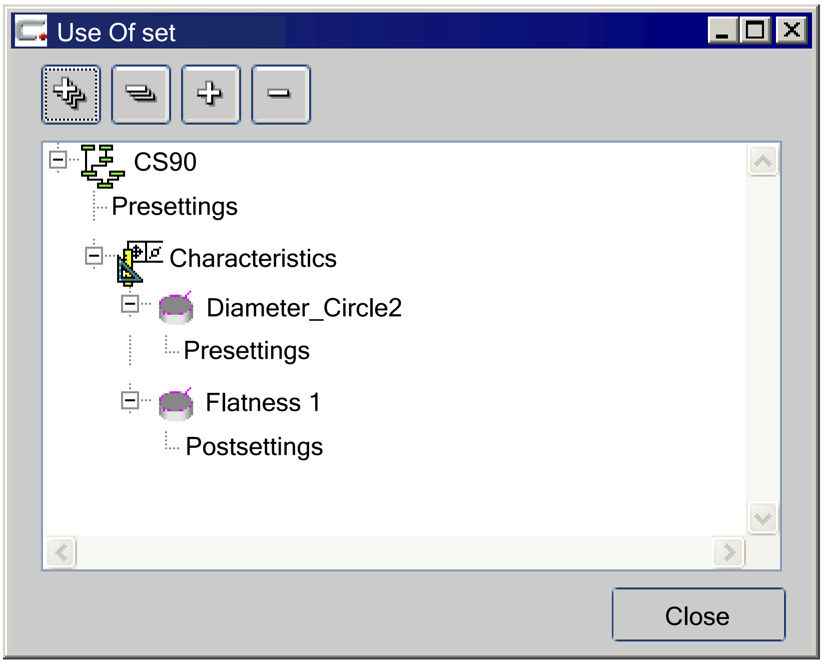
You can search the entire measurement plan for certain character strings in PCM expressions.

Select **Plan → Advanced → Search → PCM Text**.



Enter the desired character string in the **PCM Text** dialog box. Use **Function** to open the **Function and parameter list** from which you can easily take general functions and parameters as well as parameters of the current measurement plan.

By clicking **Search**, CALYPSO searches all PCM expressions of the measurement plan and shows the character strings found:



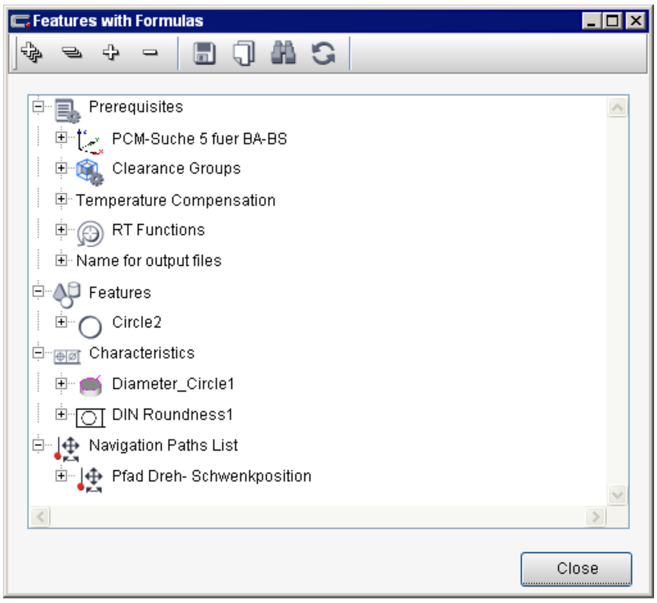
The **Use Of <pcmtext>** dialog box shows all character strings found in a hierarchically ordered tree structure. The buttons for extending and compressing all or selected branches make it possible to customize the tree structure to suit your requirements and to enable fast access to the points of interest.

Double-click the icon of the characteristic or feature in this dialog box to select the corresponding feature in the measurement plan.

Searching for features with formulas

You can look for features with formulas in the entire measurement plan.

Select **Plan** → **Advanced** → **Search** → **Features with Formulas**.



In the **Features with Formulas** dialog box, CALYPSO shows all elements containing a formula (features, characteristics, base alignments, rotary table functions, navigation paths, etc.) in a hierarchical tree structure.

The buttons for extending and compressing all or selected branches make it possible to customize the tree structure to suit your requirements and to enable fast access to the points of interest.

Double-click the icon of the characteristic or feature in this dialog box to select the corresponding feature in the measurement plan.

You have the following options:

- Click the **Save** icon to save the search results in a CSV file in any directory.
- Click the **Copy search results displayed** icon or use CTRL+C to copy the displayed structure to the clipboard and use CTRL+V or SHIFT+INSERT to insert it in another program such as Word.
- By clicking the **Search** icon you open the **Search** window allowing you to search for any character string in the expanded tree structure.
- Click the **Update** icon to refresh the display without reopening the window.

Marks in the measurement plan area

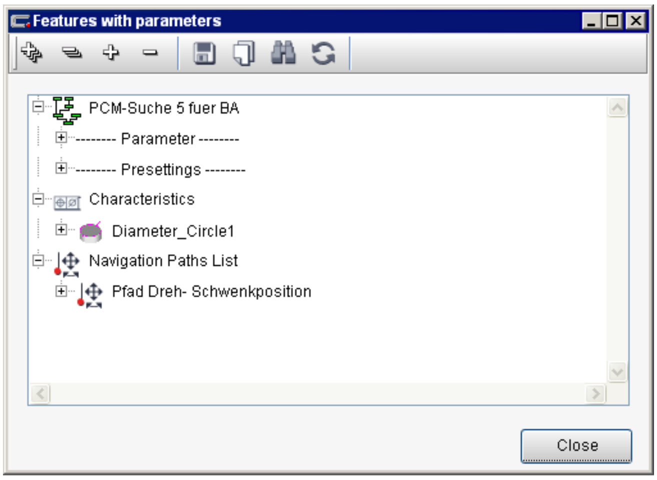
In the measurement plan area, you can select elements to which a formula is assigned.

In the **View** menu, activate the **Show features with formulas and parameters** function. In the measurement plan area, the corresponding features are marked with the letter f (formula) or fp (formula and parameter) with yellow background.

Searching for features with parameters

You can look for features with parameters (presettings and postsettings) in the entire measurement plan.

Select **Plan** → **Advanced** → **Search** → **Features with parameters**.



PCM-Suche 5 fuer BA
-------- Parameter --------
-------- Presettings --------
Characteristics
Diameter_Circle1
Navigation Paths List
Pfad Dreh- Schwenkposition

CALYPSO shows all elements containing presettings or postsettings (features, characteristics, navigation paths) in a hierarchical tree structure in the **Features with parameters** dialog box. The measurement plan parameters (presettings and postsettings) are displayed additionally..

The buttons for extending and compressing all or selected branches make it possible to customize the tree structure to suit your requirements and to enable fast access to the points of interest.

Double-click the icon of the characteristic or feature in this dialog box to select the corresponding feature in the measurement plan.

You have the following options:

- Click the **Save** icon to save the search results in a CSV file in any directory.
- Click the **Copy search results displayed** icon or use CTRL+C to copy the displayed structure to the clipboard and use CTRL+V or SHIFT+INSERT to insert it in another program such as Word.
- By clicking the **Search** icon you open the **Search** window allowing you to search for any character string in the expanded tree structure.
- Click the **Update** icon to refresh the display without reopening the window.

Marks in the measurement plan area

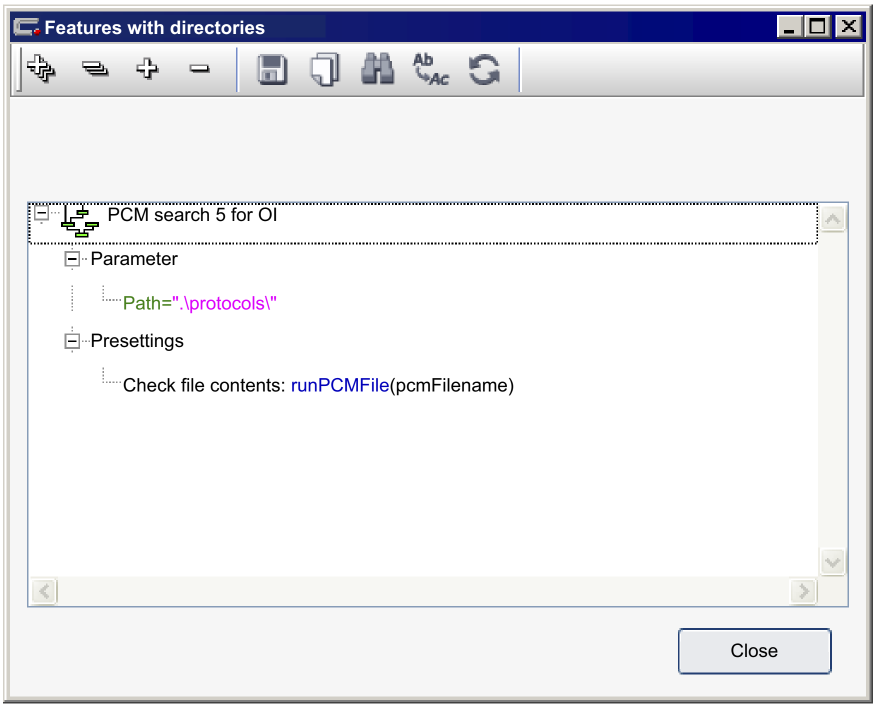
In the measurement plan area, you can select elements for which input/ output parameters are defined.

In the **View** menu, activate the **Show features with formulas and parameters** function. In the measurement plan area, the corresponding features are marked with the letter p (parameter) or fp (formula and parameter) with yellow background.

Searching features with directories

You can look for features with directories in the entire measurement plan.

Select **Plan** → **Advanced** → **Search** → **Features with directories**.



PCM search 5 for OI
Parameter
Path=".\protocols\"
Presettings
Check file contents: runPCMFile(pcmFilename)

CALYPSO shows all features and characteristics containing directory names in a hierarchically ordered tree structure in the **Features with directories** dialog box:

- Measurement plan features
  - Features
  - Characteristics
  - Global parameters
  - Presettings and postsettings of the measurement plan, features, characteristics and navigation
  - Names of output files
- External PCM files

The buttons for extending and compressing all or selected branches make it possible to customize the tree structure to suit your requirements and to enable fast access to the points of interest.

Double-click the icon of the characteristic or feature in this dialog box to select the corresponding feature in the measurement plan.

You have the following options:

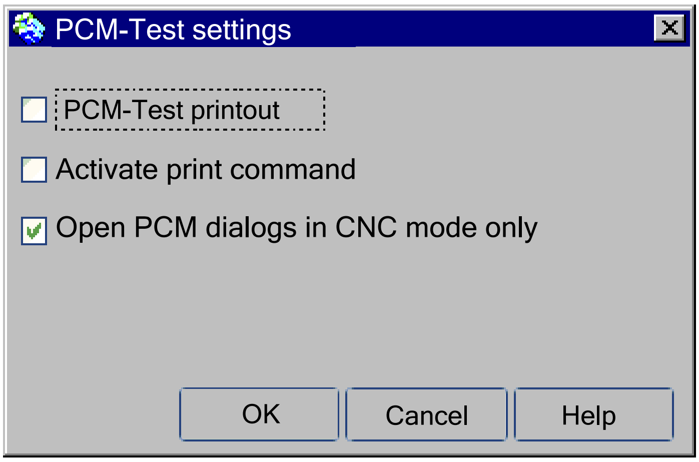
- Click the **Save** icon to save the search results in a CSV file in any directory.
- Click the **Copy search results displayed** icon or press CTRL+C to copy the displayed structure to the clipboard. It can be inserted again in another program, e.g. Word, using CTRL+V or SHIFT+INSERT.

- By clicking the **Search** icon you open the **Search** window allowing you to search for any character string in the expanded tree structure.
- By clicking the **Replace** icon you open the **Replace** window allowing you to replace any character string in the expanded tree structure by another character string.
- Click the **Update** icon to refresh the display without reopening the window.

Carrying out PCM test settings

The test settings allow you to activate different modes for the execution of PCM commands, e.g. the logging process or the execution of print commands.

1 Select **Plan** → **Advanced** → **PCM-Test settings**.
The PCM-Test settings window appears on the screen.



PCM-Test settings

□ PCM-Test printout

□ Activate print command

☑ Open PCM dialogs in CNC mode only

2 Activate **PCM Test Printout** to log the execution of each formula. Each time a value is assigned or a variable calculated, the new value is output in the default printout.
Bear in mind that **PCM Test Printout** makes the program run more slowly.

3 Tick **Activate print command** to activate the print commands in your PCM inputs.
The print command is not executed unless this check box is activated. In this way you can print the current values of the variables at any point in your measurement plan.

4 Activate **Open PCM dialogs in CNC mode only** to activate the opening of PCM dialogs exclusively in the CNC mode.

After having entered PCM functions, the functions are called and, for example, the value of PCM variables is updated. Also those PCM commands are executed that open a dialog. If this is not necessary, activate this check box.

5 Click **OK**.

The window will be closed. The settings are used the next time the measurement plan is run.

NOTE

You will be able to use CALYPSO's speed to best effect if you bear the following in mind:

– Activate the print command only for test purposes.

– Activate the PCM test printout only for test purposes.

Showing formulas in the default printout

You can have the calculated formulas, presettings and postsettings output in the default printout. This is an additional option to verify your programming.

1 Select **Resources → Characteristics Settings Editor**.
The **Measurement Plan Editor Characteristics** opens.

2 Select **Printout → Formulas in default printout** and set the value to Yes.

From now on, all formulas, presettings and postsettings will also be output in the default printout.

Examples for PCM

Overview: Examples for PCM

The examples below illustrate the versatility at your disposal using PCM with CALYPSO.

- ➤ *Using variables [➪ 1-40]*
- ➤ *Condition with alternative [➪ 1-41]*
- ➤ *Principle of parameterization [➪ 1-42]*
- ➤ *Parameterizing point recall using the list function [➪ 1-44]*
- ➤ *Loading parameter values into a curve [➪ 1-44]*
- ➤ *Defining curve using cosine function [➪ 1-45]*
- ➤ *Rechecking characteristic in case of deviation from tolerance [➪ 1-46]*
- ➤ *Parameterizing alignment [➪ 1-48]*
- ➤ *Loading a PCM file via dialog [➪ 1-48]*
- ➤ *Defining loop with linear offset [➪ 1-49]*
- ➤ *Nested loop for field (array) [➪ 1-50]*
- ➤ *Parts family and variant control [➪ 1-52]*
- ➤ *ASCII parameter file [➪ 1-53]*
- ➤ *Transferring the machine number to Q-DAS [➪ 1-54]*
- ➤ *Using the rotary table for positioning [➪ 1-56]*

Example for PCM: Using variables

This example shows how to define variables, how to access them, and how to use them in computations or other operations.

- The measurement plan contains the following variables:
  - P1 = 10
  - vector_1 = vector(10,20,30)
  - point_1 = point(10.1,20.5,30.02,0,0,1)
  - text_1= "This is a text in the text_1 variable"
- Parameter accesses to vector_1:
  - vector_1.x => return value is 10
  - vector_1.y => return value is 20
  - vector_1.z => return value is 30

- Parameter accesses to point_1:
  - point_1.x => return value is 10.1
  - point_1.y => return value is 20.5
  - point_1.z => return value is 30.02
  - point_1.nx => return value is 0
  - point_1.ny => return value is 0
  - point_1.nz => return value is 1
- Examples for calculations with variables:
  - Pnew = P1 *2. The return value is 20
  - Pnew = (vector_1.x-point_1.z)/2. The return value is -10.01
  - r = type051_n0.x/2
  - zdelta = (r1-r)/tanphi
  - norm = (nx4*nx4)+(ny4*ny4)
- Examples for output in the default printout:
  - print("Radius_1 has the value: ",radius_1)
    CALYPSO outputs the following string (example):
    Radius_1 has the value: 26.2655
  - print("r1 = ",r1,", r = ",r,",tanphi = ",tanphi)
    CALYPSO outputs the following string (example):
    r1= 23.5, r = 46, tanphi = 0.7874

NOTE

The Print command is not executed unless you have ticked the **Test printout** check box in **Plan** → **Advanced** → **PCM Test settings**.

Example for PCM: Condition with alternative

You can enter conditions with alternatives (IF - ELSE) in the input and output parameters.

The syntax is:

```
if CONDITION then
      DEFINITION1
   else
      DEFINITION2
endif
```

Meaning: If CONDITION is satisfied, DEFINITION1 is processed; if not satisfied, DEFINITION2 is processed.

DEFINITION1 and DEFINITION2 can be formulas, value assignments, functions or other conditions of your choice, always with line breaks as separators.

The example below shows how this syntax is used:

```
message("Test if with PCM")
P1 = 1
message("value is:" ,P1)
//------------------------------------------------------------
if P1 == 1 then
   message("is equal. Value was:" ,P1)
endif

if P1 < 1 then
   message("is smaller than 1: Value was:" ,P1)

else
   if P1 > 1 then
      if P1 > 5 then
         message("is greater than 5: Value was:" ,P1)
endif
message("is greater than 1: Value was:" ,P1)
   endif
endif
//-----------------------------------------------------------
test = point(1,2,3,0,0,1)
message("X" ,test.x, "Y",test.y, "Z",test.z, "nx",test.nx,
"ny" ,test.ny, "nz" ,test.nz)
if test.x == 1
then
   message("X value is:" ,test.x)
endif
```

Example for PCM: Principle of parameterization

The example below illustrates the principle of parameterization.

1 You begin by compiling an ordinary measurement plan:

- Take a perforated plate, define the base alignment at the top left front and probe twice in -Z.
- For the two probing points “point_A” and “point_B”, define the two corresponding features “Z-value_A” and “Z-value_B”.

2 Define the variables so that the positions of the probing points can be parameterized:

- Select **Plan** → **Advanced** → **Parameter**, and enter the following:

```
X_value_point_A = 10
Y_value_point_A = 25
Z_value_point_A = 0
X_value_point_B = 30
Y_value_point_B = 40
Z_value_point_B = 0
```

- Click **OK** to terminate the input.

The positions of probing points “point_A” and “point_B” can now be parameterized.

3 You do this by substituting variables for the fixed X, Y and Z coordinates:

- Open probing point “point_A”, click in the input field for the X value and select **Formula** from the context menu.
- Click **Settings** in the formula window.

The variables you defined beforehand are listed.

- Click **X_value_point_A**.

Instead of X the measurement plan now contains a variable with 10 as its parameter value.

- Repeat the entire procedure for the Y and Z values of A.

Parameters have now been substituted for the fixed X, Y and Z values of probing point “point_A”.

- Proceed in precisely the same way to substitute parameters for the fixed X, Y and Z values of probing point “point_B”.

4 Run the measurement plan.

The CMM moves to the coordinates.

If you now assign other parameters to the variables

```
X_value_point_A = 18
Y_value_point_A = 37
Z_value_point_A = 0
X_value_point_B = 44
Y_value_point_B = 23
Z_value_point_B = 5
```

and restart the measurement plan, the CMM will move to the new coordinates.

Example for PCM: Parameterizing point recall using the list function

The “Recall Feature Points” function supports the recall of any measured points of other already defined features to evaluate them in a different manner. You can recall individual points, individual paths, specific areas (specified in coordinates) or all points of one or more features.

When recalling points, you can parameterize the features to be recalled and their ranges by using the list function of PCM.

In the following example, the list of the points to be recalled is defined differently depending on the desired accuracy.

Example

```
if (Variable_Accuracy> 0.02)
   a=list("Circle1","Circle2")
else
   a=list("Circle1","Circle2","Line1","Line2","Line3","Line4")
```

Example for PCM: Loading parameter values into a curve

This example illustrates how to generate a curve from 12 individual points, using the point generator.

1 Create the following parameter file:

```
type103[1]=point(-65.386, -46.674, 139.666, -0.019, -0.022, 1.0)
type103[2]=point(-81.733, -58.343, 139.007, -0.026, -0.028, 0.999)
type103[3]=point(-98.08, -70.011, 138.154, -0.034, -0.033, 0.999)
type103[4]=point(-114.426, -81.68, 137.101, -0.043, -0.039, 0.998)
type103[5]=point(-130.773, -93.348, 135.821, -0.053, -0.046, 0.998)
type103[6]=point(-147.12, -105.017, 134.29, -0.063, -0.054, 0.997)
type103[7]=point(-163.466, -116.685, 132.499, -0.073, -0.061, 0.995)
type103[8]=point(-179.813, -128.354, 130.44, -0.084, -0.069, 0.994)
type103[9]=point(-196.159, -140.023, 128.113, -0.094, -0.077, 0.993)
type103[10]=point(-216.593, -154.608, 124.829, -0.107, -0.087, 0.99)
type103[11]=point(-220.679, -157.525, 124.122, -0.109, -0.089, 0.99)
type103[12]=point(-224.766, -160.442, 123.4, -0.112, -0.092, 0.99)
```

These lines define the 12 points from which the curve will be generated.

- If you want to include a comment for each curve point, add the following lines to the parameter file:

```
name103[1] = "type103_n1_p1"
name103[2] = "type103_n2_p2"
name103[3] = "type103_n3_p3"
name103[4] = "type103_n4_p4"
name103[5] = "type103_n5_p5"
name103[6] = "type103_n6_p6"
name103[7] = "type103_n7_p7"
name103[8] = "type103_n8_p8"
name103[9] = "type103_n9_p9"
name103[10] = "type103_n10_p10"
name103[11] = "type103_n11_p11"
name103[12] = "type103_n12_p12"
```

- You must also open the **Point Generator** dialog box (see below) and define the "`name103[index]`" variable in the **Comment** box.

2 Open the definition template of the "curve".

3 Go to **Nominal Data** and select **Parameter Data**.
The **Point Generator** window is opened.

4 Enter the following values in the **Point Generator** dialog box:

- Start Index = 1
- End Index = 12
- Increment = 1

5 Click the **Point** input field and use the context menu to open the Formula input window.

6 Enter the following variable name:
`type103[index]`
You have now entered a changeable variable name. The point generator replaces "[index]" with the current value of the loop counter in each step, starting with the start index.

7 Close the Formula Interface window.

Example for PCM: Defining curve using cosine function

You can use the point generator to define a curve by its mathematical description. This example, by way of illustration, shows how to define a curve with the aid of the cosine function. When you do this you are using implicit variable definition and value assignment by means of a formula.

Defining a curve

1 In the point generator, enter 1 as the Start Index, 50 as the End Index, and 1 as the Increment.

2 Right-click in the **Point** input field to open the Formula input window.

3 Enter the following:

```
point(index*10,50*cos(index*10),0,0,0,1)
```

This formula defines a series of 50 points with changing X and Y values, all situated on the plane Z=0 and having the same vector (0,0,1).
"point(x,y,z,nx,ny,nz)" defines a point variable. Use the "index" loop counter of the point generator as the argument in the cosine function. In each of the points the x value is index*10, the y value is 50*cos(index*10), and the z value is 0.

4 Close the Formula input window.

Example for PCM: Rechecking characteristic in case of deviation from tolerance

You can use PCM to recheck a characteristic in the event of an excursion past a tolerance limit. This is useful, for example, if the excursion is due to the presence of foreign matter. Once the foreign matter has been removed by cleaning, a repetition of the measurement will furnish the actual values.

The characteristic in this example is the roundness of a hole. If the characteristic goes out of tolerance, you want to remove foreign matter from the hole and repeat the measurement.

Task with PCM functions

1 Place a loop with 6 repetitions around the characteristic.

2 Point to the **Break Condition** field, open the shortcut menu and select **Formula...**
The **Formula** dialog box appears on the screen.

3 In the **Boolean Expression (Yes / No Test)** field, define "status_5==1" as the abortion condition and click **OK**.

4 Write the following program code into the field for the output parameters of the characteristic:

```
//
*********************************************************************************
***************
// Measure Circle hole_5 , check roundness; if out of tolerance, interrupt CNC
and request next
// task: repeat feature or measure next feature or terminate CNC run.
//
*********************************************************************************
***************
// Defining starting conditions
status_5 = 2
hole_5 = getActual("DIN Round_hole_5").actual
// If out of tolerance, open a window and inquire CNC End, Continue or Repeat
if hole_5 > 0.01 then
message("DIN Roundness of hole_5 is:" ,hole_5)
status_5 = inquire("1=CNC-End, 2=Continue, 3=Repeat. Enter a number")
   message(status_5,"Is your entry correct?", "DIN-Roundness of hole_5 is",
hole_5)
      if status_5 == 2
         then
         message("CNC run will be continued")
      status_5 = 2
      endif
endif


if status_5 == 1 then
   message("The CNC run will be terminated due to excess tolerance in hole_5.")
   cncBreak()
endif


if status_5 == 3 then
   message("The circle measurement is repeated.")
      if LOOP1 == 5 then
         message("CNC run will be terminated due to too many loop runs in
hole_5.")
         cncBreak()
      endif
endif
print("Status is",status_5, "DIN Roundness of hole_5 is", hole_5)
```

5 You can now run the measurement plan.

Example for PCM: Parameterizing an alignment

You can parameterize the angle of rotation of a system of coordinates. This entails creating the angle as a value of a variable in radian measure. You can also parameterize the offset of a system of coordinates.

In this example, the system of coordinates has to be rotated 180 or 0 degrees, depending on the value of a variable.

You accomplish this by assigning a value of 0 or 1 to a variable, for example “PartRotationActive”, in the input parameters at the respective points in the measurement plan.

1 Open the base alignment.

2 Select **Change active Base Alignment** and click **Special**.

3 In the **Special Functions** window, click **Rotate by an angle**.

4 Right-click in the input field for the angle to the context menu, open the **Formula** dialog box and enter the following:

```
3.14159265359 * PartRotationActive
```

In radian measure, 180° corresponds to the number π = 3.14159265359. The result of the formula depends on the variable PartRotationActive.

– When “PartRotationActive=0” is set, the formula’s return value is 3.14159265359 * 0 = 0°.

  The base alignment is not rotated.

– When “PartRotationActive=1” is set, the formula’s return value is 3.14159265359 * 1 = 180°.

  The base alignment is rotated 180°.

Example for PCM: Loading a PCM file via dialog

You can manually load a PCM file for the current measurement plan by clicking **Plan → Advanced → Parameter**. With the `readPCMFile` command, you can have the same function run automatically.

If you are using character string functions and polls, you can ask for the required file name in the dialog.

With the following lines, you can implement a poll for the desired parameter file and the loading of the relevant file:

```
// Select file
FILE_SELECTION = inquireList("Which file do you want to use?",
"parameter_record_1", "parameter_record_2")
// Check cancelation
if FILE_SELECTION == ""
   cncBreak()
endif
// Specify path for files
PATH = ("C:\Users\Public\Documents\Zeiss\CALYPSO\PCM_Files\")
// Load PCM file
readPCMFile(PATH + FILE_SELECTION)
```

Example for PCM: Defining loop with linear offset

Let us assume that you want to parameterize a hole pattern with linear offset in X. You can use the **Pattern** function or PCM to accomplish this task. The PCM is illustrated here. You must parameterize the X, Y and Z coordinates along with the diameter and the offset in X.

1. Select **Plan** → **Advanced** → **Parameter** and enter the following values:

   ```
   Circle_position = point(20.5,15.5,-5,0,0,1)
   OffsetX = 10
   Circle diameter = 50
   ```

2. Parameterize the circle:
   - Define the base alignment.
   - Define the circle with measurement strategy.
   - Open the feature definition template for a circle and enter parameters in the Formula input window.
   - Input field **X**: circle_position.x
   - Input field **Y**: circle_position.y
   - Input field **Z**: circle_position.z
   - Input field **D**: circle diameter

3. Define the “Diameter” characteristic and place a loop over it.
   If more than one characteristic is needed, e.g. X value, Y value, and diameter, you have to parameterize the nominals for all characteristics. Then combine the characteristics in a group and place the loop around the group.

4. Parameterize the nominal.

5. Place a loop over the characteristic:

- Start = 1
- End = 5
- Increment = 1

The next step is to update the feature with loop counter and offset parameter.

6 In the “Circle” definition template, select “Formula” from the context menu opened by right-clicking in the **X Nominal** field.
The field for the X value contains the “circle_position.x” variable. You want to increment this value by the “offsetX” value each time the loop is run.
Enter the following in the formula:

```
circle_position.x + ((LOOP1 - 1) * offsetX)
```

LOOP1 is the loop variable of the 1st loop; you can transfer it into the formula by clicking **Loop**. The start index for LOOP1 is the value from the **Start** input field for the loop.

7 Run the measurement plan.

The loop is run. The expression for the X value assumes the following values one after the other: 20.5, 30.5, 40.5, 50.5, 60.5.

Example for PCM: Nested loop for field (array)

In this example, a perforated plate is to be parameterized. The number of holes and the positions of the holes are both random. Build up the circles as an “array” (field) and use a loop in the measurement plan which you use to pass through the array index.

An array can be obtained by indexing a variable with square brackets. The array will, of course, have to be set as a parameter for the actual processing process. In this case, use the LOOP1 loop index.

1 Create the following ASCII file for PCM:

```
// Circle coordinates as points with values for x,y,z
//
CirclePos[1] = point(10,20,-5,0,0,1)
CirclePos[2] = point(15,30,-5,0,0,1)
CirclePos[3] = point(22,28.3,-5,0,0,1)
CirclePos[4] = point(40.5,30.8,-5,0,0,1)
Circle_diameter[1] = 70
Circle_diameter[2] = 50
Circle_diameter[3] = 25
Circle_diameter[4] = 22
NumberCircles = 4
```

```
// Comment strings for each circle
textCircle[1] = "This is circle A"
textCircle[2] = "This is circle B"
textCircle[3] = "This is circle C"
textCircle[4] = "This is circle D"
```

The ASCII parameter file can be generated e.g. using the Notepad editor in Windows NT.

2 Save this file as “perplate_A.para”.

3 Create a measurement plan with base alignment and a circle.
The elements of the base alignment can also be parameterized. They are omitted here for the sake of clarity.

4 Define a circle with measurement strategy.

5 Define each of the variables contained in the parameter file described above as an array (with square brackets and the LOOP1 loop variable) and read in the “perplate_A.para” parameter file.
The parameters from the file are assigned to the variables.

6 Open the feature definition template for the circle.

- In the **X Nominal** input field, enter “CirclePos[LOOP1].x”.
- In the **Y Nominal** input field, enter “CirclePos[LOOP1].y”.
- In the **Z Nominal** input field, enter “CirclePos[LOOP1].z”.
- In the **D Nominal** input field, enter “Circlediameter[LOOP1]”.
- Open the commentary field with **Formula** in the **Name/Comment** window and enter the “textCircle[LOOP1]” text parameter.

7 Define the characteristics X value, Y value and diameter. The nominal values are parameterized using **Formula**.

8 Combine the characteristics in a group.

9 Place a loop over the group:

- Start = 1
- End = NumberCircles
You can import the “NumberCircles” variable into the **End** field using the Formula input window.
- Increment = 1.

10 Start the measurement plan.

Depending on the content of the parameter file, you can use this measurement plan to measure different numbers of circles with different positions and diameters.

Example for PCM: Parts family and variant control

PCM enables you to measure different variants of a workpiece using a single measurement plan.

Let us assume, for example, that a workpiece has two variants, A and B. Variant B has three extra holes with a diameter of 0.39 in.

1. Create a complete measurement plan, including the three holes.
2. Combine all characteristics of the three holes in the group called “Additional hole pattern variant B”.
   Selection of the variant is controlled by means of a PCM parameter as a switch.
3. Select **Plan** → **Advanced** → **Parameter** and enter the following:

```
// Measure additional hole pattern variant B when parameter is 1
MeasureAdditionalHolePatternActive = 1
```

4. Highlight the “Additional hole pattern variant B” group and set a condition:
   - Select **Condition** from the context menu.
   - Click in the input box for the condition to open the context menu and open the Formula input window.
   - In the Formula input window, click **Settings**.
   - Confirm the **MeasureAdditionalHolePatternActive** variable.
   - Insert “== 1” as an additional entry.

   The condition is now defined: the “Additional hole pattern variant B” group is measured only when the “MeasureAdditionalHolePatternActive” variable is equal to 1.
5. Run the measurement plan.
   The “Additional hole pattern variant B” group is measured.
6. Select **Plan** → **Advanced** → **Parameter** and enter the following:

```
// Measure additional hole pattern variant B when parameter is 1
MeasureAdditionalHolePatternActive = 0
```

7. Run the measurement plan again.

The “Additional hole pattern variant B” group is not measured in this run.

Example for PCM: ASCII parameter file

Example of an ASCII parameter file (*.PARA file):

```
// Text-Parameters -------------------------------------------
who = "File generated by ZEISS-IMT Training Center"
nameOfCircle_1 = "Circle_1"
nameOfCircle_2 = "Circle_2"
nameOfCircle_3 = "Circle_3"
text1 = "type034_n2_p2"
nextText = "This is in Front"

// Text-Parameters as an array ---------------------
name101[1] = "type101_n1_p1"
name101[2] = "type101_n2_p2"
name101[3] = "type101_n3_p3"
name101[4] = "type101_n4_p4"
name101[5] = "type101_n5_p5"
name101[6] = "type101_n6_p6"
name101[7] = "type101_n7_p7"
name101[8] = "type101_n8_p8"
name101[9] = "type101_n9_p9"
name101[10] = "type101_n10_p10"
```

```
// Numeric-Parameters ------------------------------
Partnumber = 10
PartRotationAktiv = 1
CREATION_DATE = 950516
CREATION_TIME = 0000
DESIGN_DATE = 950516
NUMBER = 772209709750
PART_HEIGHT = 316.000
PLACE = 2
X_Circle_1 = 10
Y_Circle_1 = 20
Z_Circle_1 = 15
D_Circle_1 = 40
X_Circle_2 = 250
Y_Circle_2 = 40
Z_Circle_2 = -15
D_Circle_2 = 25
X_Circle_3 = -12
Y_Circle_3 = -20
Z_Circle_3 = 50
D_Circle_3 = 12

// Numeric-Parameters as an array type vector -------------------
type086[1] = vector( 10.0, 0.0, 0.0 )
type086[2] = vector( 0.0, -20.0, 0.0 )
type086[3] = vector( 30.0, 20.0, 10.0 )
type086[4] = vector( 17.5, 13.8, 50.0 )

// Numeric-Parameters as an array type point --------------------
CurveFront[1] = point( 216.023, 0.0, 69.698, 0.984, -0.0, 0.176 )
CurveFront[2] = point( 206.34, 0.0, 102.988, 0.916, -0.0, 0.401 )
CurveFront[3] = point( 180.14, 0.0, 141.726, 0.707, -0.0, 0.707 )
CurveFront[4] = point( 130.593, 0.0, 181.254, 0.575, -0.0, 0.818 )
CurveFront[5] = point( 107.046, 0.0, 197.639, 0.57, -0.0, 0.822 )
CurveFront[6] = point( 80.825, 0.0, 216.02, 0.557, -0.0, 0.83 )
```

Example for PCM: Transferring the machine number to Q-DAS

This example illustrates how to transfer the machine number to the Q-DAS interface using PCM commands. The following commands are written into the input parameters:

```
// EXAMPLE OF PCM-COMMANDS TO TRANSFER THE MACHINE-ID TO Q-DAS
// TO ACTIVATE PUT IT INTO THE PRE-PARAMETER OF THE INSPECTION
// OR INTO THE FILE 'inspection_start_pcm.txt'
// IN THE DIRECTORY OF THE TESTPLAN

// Testassignment (normally from FACS)
OP10_machine="4711"
OP20_machine="0815"
OP30_machine="007"
OP40_machine="abc"

// Function for calculationg machine id for Q-DAS
defineFunctionStart("CALC_MACH_NR")
OP_NAME=getCFAttribute("u_k0053operation")
PARA_NAME="OP"+OP_NAME+"_machine"
MACH_ID=getParameterNamed(PARA_NAME)
defineFunctionEnd("MACH_ID")

// Function for calculationg machine id for Q-DAS
defineFunctionStart("CALC_MACH_NR2")
OP_NAME=getCFAttribute("u_k0053operation")
PARA_NAME="OP"+OP_NAME+"_machine"
MACH_ID=getParameterNamed(PARA_NAME)+"m"
defineFunctionEnd("MACH_ID")

// Generate new column 'u_machine_id' in table file
generateTableColumn("u_machine_id","CALC_MACH_NR")
generateTableColumn("u_2machine_id","CALC_MACH_NR2")

// Activate multiQdasOutput and set columnID for file splitting
outputMultiQdas("u_k0053operation")

LISTE=readListFile("liste.txt")
// LISTE=list("Radius Tangential","Radius Hüll",
// "Gruppe: best fit of bore pattern")
addCF(LIST)
// addME("PlaFroTop", "PlaBack")
// setRunID("Run3er", "RunGAU PFER")
```

The central “outputMultiQdas” command generates additional columns in the table file:

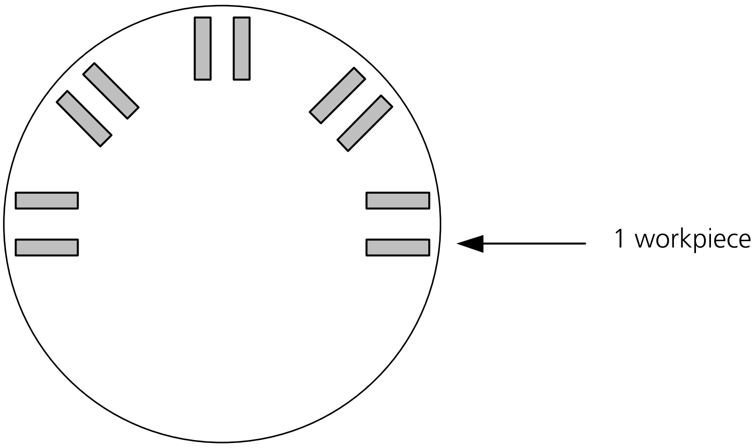
AB	AC
u_machine_id	u_2machine_id
0815	0815m
abc	abcm
007	007m

Example for PCM: Using the rotary table for positioning

You can use the rotary table as positioning unit for the measurement of equal workpieces. You must create the positioning procedure for a workpiece only once and must change the rotary table position and the workpiece position, if necessary, via loops.

Example of application

This graphic shows a case in which both options are used: the workpiece position is changed once via an offset for the rotary table position and the measurement plan is processed five times using different rotary table positions.



Thus, the number of equal workpieces which can be measured in this way is double the number of parameterized rotary table positions.

Procedure

If you wish to use the rotary table to measure several equal workpieces in one measurement plan, you must proceed as follows:

Defining the rotary table position

You prepare the entry of the rotary table position in the presettings. To do this, use the *rtPositionOffset* PCM function.

NOTE

You can also enter a formula as the argument of the PCM function. For example, the angle can be calculated as a function of a part index which is prompted every time you start. You can also predefine the angle in AutoRun.

RT home position

If you entered the *rtPositionOffset* PCM function in a measurement plan, you must home the RT prior to the CNC run.

The offset angle is saved in the measurement plan and will keep its value also after deleting the function. Every time when changing to another measurement plan or when removing the *rtPositionOffset* PCM function from this measurement plan or adding it again, you must home the RT manually. By doing this, the internal value is reset.

NOTE

It is not absolutely necessary to home the RT between the individual runs of a measurement of series.

PCM quick reference

Overview: PCM quick reference

You need the following information in order to program with PCM:

- ➤ *PCM syntax [➪ 1-59]*
- ➤ *Variables in PCM [➪ 1-59]*
- ➤ *Arithmetic and comparative operators in PCM [➪ 1-60]*
- Overview of PCM functions:
    - ➤ *Mathematical Functions [➪ 1-61]*

      Contains the syntax for mathematical functions such as sine, cosine, etc.
    - ➤ *Character String Functions [➪ 1-64]*

      Contains the syntax for functions using character strings such as asc, chr, len, etc.
    - ➤ *Input and Output [➪ 1-66]*

      Contains the functions that control screen input and output.
    - ➤ *File Commands [➪ 1-70]*

      Contains the functions required for working with files and directories.
    - ➤ *Measuring-specific and measurement plan-specific functions [➪ 1-82]*

      Contains the functions that process measured values generated by a measurement plan run and that read measurement plan-specific data such as stylus properties.
    - ➤ *CMM-specific functions and travel commands [➪ 1-127]*

      Contains the functions for querying CMM-specific data and for controlling the CMM.
    - ➤ *System Commands [➪ 1-136]*

      Contains the system calls.
    - ➤ *Custom Printout [➪ 1-140]*

      Contains the functions for activating and deactivating the custom printout and for determining the formats.
    - ➤ *Conditions/Loops [➪ 1-142]*

      Contains the syntax for conditions and loops.
    - ➤ *Output in printout [➪ 1-146]*

      Contains the functions for the definition of the output in printouts.
- Overview of the ➤ *names of external PCM files [➪ 1-148]*

PCM syntax

Like the programming languages Basic, C, Fortran and so on, PCM uses a functional syntax with the following rules:

- Definitions and value assignments as follows:

  variable name=value

- Blanks are not permitted in names and formulas; the syntax is case-sensitive, so it distinguishes between uppercase and lowercase letters.
- Multiplication and division precede addition and subtraction when formulas are resolved.
- The decimal separator is the point (example 3.85).
- In functions, the function parameters (the arguments) are in parentheses and separated by commas.
- In commands (procedure calls), the function parameters (the arguments) can be omitted. Example: getActual().
- Comments can be entered in each line: everything following the “//” string is ignored.

Variables in PCM

There are six types of variables in PCM. The type of variable is defined implicitly by the value assignment:

Variable type	Examples of value assignments

number	e = 2.71828
	Variable2 = 2.0
	P1 = 80

vector	Axis = vector(10,12,0)
point	CylinderB = point(-10,12.5,0,0,0,1)

string	Text_1 = "Circle"
	Text_2 = "Enter the number:"

Boolean	In the Multiple Printouts mode = false Reset = getStartSetting("clearOldRes")

list	coll=list("Circle1","Circle2","Circle3")
	range=list(name1,name2,name3,name4)

Arrays

Arrays can also be defined with variables using special value assignments with square brackets. An array consists of several variables of the same type. Example:

```
array[1] = point(-10,12.5,0,0,0,1)
array[2] = point(-12,12.5,0,0,0,1)
array[3] = point(-14,12.5,0,0,0,1)
array[4] = point(-16,12.5,0,0,0,1)
```

In this case, “array” is an array consisting of 4 variables of the point type.

The array index in square brackets can also be specified by a variable (of type number, as an integer). You can therefore define an array of any length in a single loop instruction. Example:

```
for I = 1 to numberTeeth
type[I] = inquireNumber("Which tooth type is in position number ",I)
next I
```

You can gain direct access to a specific element in an array by specifying the array index. Example:

```
Length = output values[4]
```

Arithmetic and comparative operators in PCM

You can use the following arithmetic operators in PCM:

Operator	Result
+	Sum
-	Difference
*	Product
/	Quotient
**	Power

You can use the following comparative operators for conditions in PCM:

Operator	Result
<	Truth value of “a < b”
>	Truth value of “a > b”

Operator	Result
<>	Truth value of “a not equal to b”
==	Truth value of “a = b”
>=	Truth value of “a ≥ b”
<=	Truth value of “a ≤ b”

Mathematical functions in PCM

Angle conversions

You can use the following functions to convert angle values in PCM:

Function	Result
rad(DegreeAngle)	Radian value from degree angle
deg(RadianAngle)	Angular degree (decimal) of radian angle
angle (angular degree)	Angular degree (decimal) of angular degrees

The angular degree is indicted in

```
degrees, minutes, seconds
```

Trigonometric Functions

PCM supports the following trigonometric functions:

Function	Result
sin(DegreeAngle)	Sine value
sinRad(RadianAngle)	Sine value
cos(DegreeAngle)	Cosine value
cosRad(RadianAngle)	Cosine value
tan(DegreeAngle)	Tangent value
tanRad(RadianAngle)	Tangent value
arcsin (value)	Arc sine in degrees
radArcsin (value)	Arc sine in rad
arccos (value)	Arc cosine in degrees
radArccos (value)	Arc cosine in rad
arctan (value)	Arc tangent in degrees
radArctan (value)	Arc tangent in rad
arctan2 (value1,value2)	Arc tangent from the quotient value1/value2 in degrees

Function	Result
radArctan2 (value1,value2)	Arc tangent from the quotient value1/value2 in rad

Logical operators

CALYPSO supports the following logical operators:

Operator	Result
(Expression1) and (Expression2)	Logical AND of the truth values of both expressions
(Expression1) or (Expression2)	Logical OR of the truth values of both expressions
not (expression)	Logical negation of the truth value of the expression

Examples:

If var1 = 0 and var2 = -7, then

- (var1=0) and (var2>0) = false
- (var1=0) or (var2>0) = true
- not (var2>0) = true

If t=true, f=false, and i=9, then

- bed1 = (i>1) and (i>10) = false
- bed2 = t and f = false
- bed3 = t and (i>10) = false
- bed4 = true or (i>10) = true

Bit operations

CALYPSO supports the following bit operations on integral values:

Operator	Result
bitAnd(number1,number2)	Bitwise AND of the double representation of the two numbers
bitOr(number1,number2)	Bitwise OR of the double representation of the two numbers
bitXor(number1,number2)	Bitwise excluding OR of the double representation of the two numbers

Examples:

- bitAnd(12,4) = 4
- bitOR(12,4) = 12
- bitXor(12,4) = 8

Other functions

Other mathematical functions are available:

Operator	Result
squared(value)	Value squared
sqrt(value)	Square root of value
exp(value)	e^{value}
ln(value)	Natural logarithm of value
log(value)	Common logarithm of value
mod(value1,value2)	Value1 modulo value2
int(value)	Integral proportion of value
abs(value)	|Value| (=absolute value of the value)
round(value[,figures])	Value, rounded to the given number of places; if specification of number of places omitted: 0 places
sign([value1,]value2)	Value1 multiplied by sign of value2 (default of value1 = 1): - value1, if value2 < 0 + value1, if value2 ≥ 0
signWith-Zero([value1,]value2)	Value1 multiplied by sign of value2 (default of value1 = 1): - value1, if value2 < 0 0, if value2 = 0 + value1, if value2 > 0
max(value1[,value2[, ...]])	Maximum of the indicated values
min(value1[,value2[, ...]])	Minimum of the indicated values
ord(BooleanValue)	1 for true0 for false

Examples:

squared(3) = 9

sqrt(225) = 15

exp(0) = 1

ln(1) = 0

log(10000) = 4

mod(22,8) = 6

int(34.5674) = 34

abs(-35.335) = 35.335

round(35.335,2) = 35.34

sign(-12,-34) = 12

sign(-12) = -1

max(1,2,5,8,3) = 8

Generating functions

The following functions generate geometric objects:

Function	Result
point(x,y,z[,nx,ny,nz])	Point with the x, y, z coordinates and (optionally) the normal vector (nx,ny,nz)
vector(x,y,z)	Vector with the coordinates x, y and z
calculatePointOnHelx(r,index,g,tw,kw)	Points on the cylinder's and cone's lateral surface or tapered thread with radius r, lead g, trapezoid angle tw and cone angle kw.

Character string functions in PCM

You can use the following character string functions and operations in PCM:

Function	Result
character string1 + character string2	Character string3 of the following characters of character string1 and character string2
asc(character)	ASCII code (number) of specified character
chr(number)	Character with the given ASCII code
cr()	Line Break
format(number)	Character string consisting of the figures and characters of the number
formatL (Number [,Total Number Places [,NumberDecimalPlaces]])	The number as a rounded decimal number with the specified decimal places in the form of a character string with the minimum length “Places” (decimal point is not considered), left-justified format
formatR(Number [,Total Number Places [,NumberDecimalPlaces]])	The number as a rounded decimal number with the specified decimal places in the form of a character string with the minimum length “Places” (decimal point is not considered), right-justified format
inStr([startIndex,] character string1, character string2)	Position of the first occurrence of character string2 within character string1 after the startIndex-th character or the first character
len(CharacterString)	Length of character string

Function	Result
mid(character string,startIndex[,length])	Character string, consisting of the characters from the startIndex-th character to the end of the character string or to the (startIndex +length-1)-th character
qm()	Represents double inverted commas (quotes) as character string.
strElement(n,char,character string)	The n-th element of the character string, when char is regarded as a separator between the elements
subStr(character string,startIndex,endIndex)	Character string, consisting of the characters from the startIndex-th character to the endIndex-th character
text(value1[,Value2,Value3])	Character string from all transferred parameters
val(CharacterString)	Number represented by the character string

Examples:

"diameter" + text(1) = "diameter1"

asc("8") = 56

chr(111) = "o"

format(3278,45) = "3278,45"

formatL(12345.6789,10,2) = "12345.68" (three blanks)

formatL(12345.67,10,4) = "12345.6700" (one blank)

formatR(12345.67,10,6) = "12345.670000" (length here > 10)

val("3278,45") = 3278,45

len("This is a character string") = 26

`qm()+"c:\zeiss\calypso\modul 4\hugo.bat"+qm()` is interpreted as character string “`"c:\zeiss\calypso\modul 4\hugo.bat"`”

inStr(3,"Position","o") = 7

inStr("Position","o") = 2

mid("PCM functions",1,3) = "PCM"

strElement(4,",","hello,here,we,are,again") = "are"

strElement(4,"e","here,we,are,again") = "ar"

subStr("This is a test",4,9) = "s is a"

PCM functions: Input and output

confirm

Displays the **Question !** window on the screen and shows true after clicking **Yes** and false after clicking **No**. The syntax is:

```
confirm("question")
```

## display

Outputs a text in a screen window. The CNC run is not interrupted. Additionally, certain parameters can be used to suppress and reenable the execution of the `display` commands. The syntax is:

```
display(parameter)
```

parameter	Function
`Text`	Opens a new **Display** window for text output.
`"#displayCLOSE"`	Closes all **Display** windows.
`"#displayOFF"`	Ignores additional `display` commands until the next `display("#displayON")` command. Subsequently, all control outputs are deactivated with one line.
`"#displayON"`	Switches the consideration of the `display` commands on again.

Examples:

```
display("Group A")
```

Displays the "Group A" text on the screen.

```
para="Group B"
```

```
display(para)
```

Displays the "Group B" text on the screen.

inquire

Corresponds to the **inquireNumber** command, but is supported by PCM only for the sake of upwards compatibility. The syntax is:

```
variable name = inquire("Dialog text for poll")
```

inquireList

The **inquireList** command can be used to create a menu for querying a character string. The syntax is:

```
variable name = inquireList("Menu title","Menu item1","Menu item2",...,"Menu
itemn")
```

The command causes the appearance of a menu with the given title line on the screen. As a value, the variable is assigned the character string of the menu item which has been selected with a mouse click or the arrow keys and confirmed with OK.

Examples:

```
NameForPrintoutheader = inquireList("Name for printout header","Test print-
out","Default printout",)
```

inquireNumber

The **inquireNumber** command can be used to create a dialog for querying a numeric value. The syntax is:

```
variable name = inquireNumber("poll text for workpiece number")
```

or

```
variable name = inquireNumber("line1"[,cr()],"line2"[,cr()],...,"lines")
```

Here the “cr()” optional elements each cause a line break on the screen.

Example 1:

```
1 = inquireNumber("Enter 1 to continue measuring")
if P1 == 1 then
   message("You have entered 1, so I will continue measuring")
else
   if P1 <> 1 then
      message("You do not want to continue measuring!!")
   endif
endif
```

Example 2:

```
number = 10
P1 = inquireNumber("Last value was: ",number,cr(), "Enter new number:")
message("You have entered the number:",P1)
```

Example 3:

```
type = inquireNumber("Cone tooth = 1", cr(),
         "Cylinder form tooth = 2", cr(),
         "Long cylinder tooth = 3", cr(),
         "Cylinder form tooth = 4", cr(),
         "Please enter number")
```

inquireParameterList

If the **inquireParameterList** command is contained in the presettings or postsettings, the user can open a dialog box and enter the parameters in it using a list. The syntax is:

```
inquireParameterList("p1","k1",...,"pn","kn")
```

All variables (p1 to pn) that are to be entered via the list must be contained in the brackets together with comments k1 to kn.

To open the dialog box, the user must highlight the line containing the command and select **Calculate** in the context menu.

inquirePasswordText

Opens the dialog box for entering a text. The entered text is masked (in the form of asterisks). The function supplies the return value of the entered text. The syntax is:

```
inquirePasswordText("dialogtitle")
```

The character string entered for `dialogtitle` is written in the title bar of the dialog box.

Example:

```
enteredText = inquirePasswordText("password:")
```

While the command is being processed, a dialog box with the title “Password:” opens. The character string entered is written in the `enteredText` variable.

inquireText

The **inquireText** command can be used to create a dialog for querying a character string. The syntax is:

```
variable name = inquireText("text line")
```

or

```
variable name = inquireText("text line1"[,cr()],"text line2"[,cr()],...,"text
lines")
```

Here the “cr()” optional elements each cause a line break on the screen.

Examples:

```
string1 = inquireText("Enter the name")
```

list

Creates a list from the transferred features. This list is required for the formula input for Recall and Recall Feature points. Access to the features is not possible. The syntax is:

```
list(Feature1,Feature2,Feature3,...)
```

Example: `coll=list("Plane1","Plane2","Plane3")`

A list of the specified texts is stored in the “coll” variable.

message

Outputs the current values of one or more variables. The syntax is:

```
message(variable,variable,variable,...)
```

Inserting “cr()” instead of a variable causes a line break on the screen.

Example:

```
message("The value of variable P1 is: ", P1, cr(), "The value of variable P2 is: ",
P2)
```

print

Prints the current values of one or more variables in the printout.

The syntax is:

```
print(variable;variable;variable;...)
```

NOTE

The print command will not be active unless you have ticked the **Activate print command** check box after selecting **Plan → Advanced → PCM Test Settings** in the **PCM Test Functions** window. This means you can use the command for test purposes.

Examples:

```
print("Print this dialog text in the printout")
```

```
print("Print the value of variable P1 in the printout"; P1)
```

redrawCAD

Recalculates all features and updates the CAD window. The syntax is:

```
redrawCAD()
```

PCM functions: File commands

addToFile

Adds a line to a file. If the file does not exist, it will be specially created. The syntax is:

```
addToFile(file name,value1,...)
```

Whenever the file name is used without path name, this function will access the directory of the current measurement plan. If the name of the drive is missing, the current directory will be used.

Example: `addToFile(wd+"\info.txt","line",1)`

Here wd is a variable that contains a path specification.

copyFile

Copies a file, i.e. saves a copy of the file under a name to be specified. The syntax is:

```
copyFile("file name1","file name2")
```

Whenever the file name is used without path name, this function will access the directory of the current measurement plan. If the name of the drive is missing, the current directory will be used.

NOTE

If a file with the name “file name2” already exists, this will be overwritten.

Example: `copyFile("test.txt","test2.txt")` creates a copy of `test.txt` in the current measurement plan directory and saves it under the name `test2.txt`.

deleteFile

Deletes the specified file. The syntax is:

```
deleteFile(file name)
```

Whenever the file name is used without path name, this function will access the directory of the current measurement plan. If the name of the drive is missing, the current directory will be used.

directoryPath

Returns the path in the file system for the specified directory. The syntax is:

```
directoryPath(directory name)
```

The following values are possible for the `Directory name` parameter:

Value	Meaning
`actuals`	The directory for actual values.
`actualInspection`	The directory of the current measurement plan.
`applicationData`	The user directory. Up to CALYPSO 5.4: *\home\om* From CALYPSO 5.6: installation directory (e.g. *C:\Users\Public\Documents\Zeiss\CALYPSO\*).
`defaultInspections`	The default directory of measurement plans (*\workarea\inspections*).
`defaultResults`	The default directory of results. (*\workarea\results*).
`formplott`	The directory of CALYPSO form plots.
`inspections`	The user-defined directory of measurement plans.
`programFiles`	The directory of program data. Up to CALYPSO 5.4: *\opt\om* From CALYPSO 5.6: installation directory (*C:\Programs\Zeiss\CALYPSO v.v*, *v.v* being the current version number).
`protform`	The directory of printout templates.
`results`	The user-defined directory of results.
`workarea`	The *workarea* directory.

exportCharacteristicsHierarchy

Outputs a hierarchically structured list of all characteristics contained in a measurement plan to an ASCII file called `cfHierarchy.txt` in the current measurement plan directory. This file can be used to define selection lists (e.g. for FACS). The syntax is:

```
exportCharacteristicsHierarchy()
```

One characteristic per line is written to the file. For hierarchy representation, the name of the characteristic is preceded by one blank per group depth.

NOTE

In combination with the automatic PCM file call function (see below), writing occurs each time before the measurement plan is saved.

exportPoints

Exports actual values and, if necessary, nominal values and deviations from a feature into a file. The syntax is:

```
exportPoints("feature_name","path_and_file_name")
```

The format and scope of output are defined in the Export Points dialog box for every individual feature. The default setting is as follows: text file, actual values + normal vectors, base alignment, contact points.

The path and file name can be specified absolutely or relative to the current measurement plan directory. Examples:

```
exportPoints("feature_name","geoactuals\Circle1")
```

```
exportPoints("feature_name","c:\temp\points.txt")
```

fileExists

Checks whether a file exists in a certain path and returns a Boolean value. The syntax is:

```
fileExists("path_and_filename")
```

The path and file name can be specified absolutely or relative to the current measurement plan directory. Examples:

```
fileExists("geoactuals\Circle1")
```

```
fileExists("c:\temp\points.txt")
```

generateTableColumn

Writes additional columns in the characteristics table file (*chr.txt). The contents of the corresponding field results from the return value of the `function name` function. The syntax is:

```
generateTableColumn(column ID,function name)
```

Within the `function name` function, you can use getCFAttribute() to access attributes of the current characteristic.

getActualInspectionDir

Returns the directory of the current measurement plan. The syntax is:

```
getActualInspectionDir()
```

getWD

Returns the current directory. The syntax is:

```
getWD()
```

Example: `wd = getWD()`

readCTFile

Imports a CT dataset (only with the Metrotomography option). The syntax is:

```
readCTFile(file name)
```

readListFile

Creates a list loaded from an ASCII file. The syntax is:

```
readListFile(file name)
```

The values in the ASCII file are read line by line and summarized in a list.

Whenever the file name is used without specifying a path, CALYPSO will first search the directory of the current measurement plan and then the general directory for measurement plans.

Example: Definition of the scope of measurement via a file. The created list can be used directly for the *addCF* PCM function.

```
THE_LIST=readListFile(scope4711.txt)
```

```
addCF(THE_LIST)
```

readPCMFile

Reads in a PCM file. The syntax is:

```
readPCMFile(file name)
```

Whenever the file name is used without path name, this function will access the directory of the current measurement plan. If the name of the drive is missing, the current directory will be used.

The new parameters which are read in will, in each case, overwrite the current parameters of the same name.

NOTE

The parameters which were available before the CNC run will only be temporarily overwritten. The original start setting will be used for the next CNC run.

The file name can also be entered in the form of several parameters. The parameters will, depending on the type, be put together to form a string.

Example 1:

```
readPCMFile("testparameter.para ")
```

Example 2:

```
P1 = "c:"
```

```
P2= "zeiss\calypso\ "+"test.para "
```

```
readPCMFile(P1,P2)
```

Example 3:

```
readPCMFile("c: ", "zeiss\calypso\ ", "test.para ")
```

readUserAttributes

Reads the characteristic attributes from an external ASCII file. The syntax is:

```
readUserAttributes(file name)
```

The characteristic attributes will then be available for further use in CALYPSO.

The file is a table file. It contains lines and columns separated by tabulator.

- The first line contains the column headings (1st column: identifier, 2nd to nth column: keyword of the respective characteristic attribute).
- Any further line contains the characteristic name in the first column and the values of the corresponding attributes in the other columns.

NOTE
Save in simple ASCII format a file that has been edited in a spreadsheet program and use tabulators as separators.

NOTE
If you also want to edit the imported characteristic attributes in CALYPSO, the “userattributes.ini” configuration file must contain the corresponding entries with the same keywords (see User-defined characteristic attributes in the operating instructions for the base module.

renameFile

Renames a file. The syntax is:

```
renameFile("file name old","file name new")
```

Whenever the file name is used without path name, this function will access the directory of the current measurement plan. If the name of the drive is missing, the current directory will be used.

Example: `renameFile("test.txt","Test2.txt")` renames the `test.txt` file `test2.txt`.

runPCMFile

Processes a text file with PCM commands and returns a numerical value. The syntax is:

```
runPCMFile("filename")
```

Whenever the file name is used without path name, this function will access the directory of the current measurement plan. If the name of the drive is missing, the current directory will be used.

The file must contain a normal PCM syntax.

NOTE
The function can only be called if the PCM Technology option is available.

After a successful call, the function returns 0 as result, in case of an error, the result will be 9999. If the file is not available, the CNC run will be stopped.

Example 1:

```
res=runPCMFile("C:\temp\PCMTest.txt")
```

Processes the PCMTest.txt file in the specified path, res is 0 if the file exists, if not, the result is 9999.

Example 2:

```
res=runPCMFile("PCMTest.txt")
```

Processes the *PCMTest.txt* file in the measurement plan directory. The value of `res` is 0 if the file exists, otherwise the value is 9999.

saveViewToFile

This function is used to save a CAD view stored via **CAD → View → Save View** as jpeg file under the name `view_name`. The syntax is:

```
saveViewToFile(view name,file name)
```

selectFileDialog

Opens a file selection dialog and returns the directory and file name.

The syntax is:

```
selectFileDialog([directory name)[,filter[,dialog text[,return mode]]]]
```

Meaning of the parameters:

Parameter	Meaning
`directory name`	Real or symbolic directory name. If blank, then the current measurement plan directly will be used.
`filter`	Optional: filter for valid entries.
`dialog text`	Optional: title bar of of the dialog window.

Parameter	Meaning
`return mode`	0 or blank: complete path with directory and file name 1: File name only 2: Directory name only 3: File name as the object

The following values are possible for the `Directory name` parameter:

Value	Meaning
`actuals`	The directory for actual values.
`actualInspection`	The directory of the current measurement plan.
`applicationData`	The user directory (e.g. *C:\Users\Public\Documents\Zeiss\CALYPSO\*).
`defaultInspections`	The default directory of measurement plans (*\workarea\inspections*).
`defaultResults`	The default directory of results. (*\workarea\results*).
`formplott`	The directory of CALYPSO form plots.
`inspections`	The user-defined directory of measurement plans.
`programFiles`	The directory of the program data (*C:\Programs\Zeiss\CALYPSO v.v*, *v.v* being the current version number).
`protform`	The directory of printout templates.
`results`	The user-defined directory of results.
`workarea`	The *workarea* directory.

Example: The following code lines return the path name of the selected PDF file in the directory *C:\aaaaa*:

```
PATH="C:\aaaaa"
FILTER="*.pdf"
DIALOG="Please select file"
MODE=0
SELECTION=selectFileDialog(PATH,FILTER,DIALOG,MODE)
```

Possible queries, for return mode 3 SELECTION.head SELECTION.tail SELECTION.asString

waitForFile

Stops the run until one of the following events occurs:

– A “File name” file is available

– A set waiting time has been reached

– The system generates the message “Directory does not exist”

The syntax is:

```
waitForFile(file_name[,message,flag1,timeout,report text,flag2])
```

Meaning of the parameters:

Parameter	Meaning
`file name`	File name with absolute or relative path
`message`	This text is displayed on the screen during the waiting time.
`flag1`	0: Do not show file name 1: Show file name
`timeout`	Waiting time in seconds
`report text`	Message in the report when the waiting time has been reached.
`flag2`	0: No check whether directory exists 1: Check whether directory exists

The return values have the following meaning:

Return value	Meaning
0	File is available
1	Waiting time has been reached
2	Directory does not exist

writeActualsToFile

Saves filtered actual values with stylus radius and status in the specified file. For each line in the file, the coordinates and the nominal vector of the point as well as the stylus radius and the status are included. A status not equal to 0 means “outlier”.

The syntax is:

```
writeActualsToFile(["characteristic",][true|false,[alignment,]] "file_name")
```

The function supplies “true” as soon as saving is completed, otherwise the function supplies “false”.

- The name of characteristic is optional. If no characteristic name is indicated, the current characteristic will be used.
- The alignment is optional. If no alignment is specified, the base alignment will be used.
- `writeActualsToFile(["characteristic",]"file_name")` exports stylus center point coordinates.
- `writeActualsToFile(["characteristic",][true|false,]"file_name")` exports surface points in the case of “true” and, in the case of “false”, stylus center point coordinates.
- For the Curve Form characteristic, only the characteristic name and the file name are specified. The points are output as surface points in the coordinate system of the curve.

Examples:

```
writeActualsToFile("diameter1",true,"c:\test.txt")
```

exports surface points.

```
writeActualsToFile("diameter1","c:\test.txt")
```

exports stylus center point coordinates.

```
writeActualsToFile("diameter1",false,"c:\test.txt")
```

exports stylus center point coordinates.

```
writeActualsToFile("Kurvenform1","c:\test.txt")
```

exports surface points of the curve's coordinate system.

writeActualsToVDA

Saves filtered actual values in VDA format in the specified file.

The syntax is:

```
writeActualsToVDA(["characteristic",][true|false,[alignment,]] "file_name")
```

The function supplies “true” as soon as saving is completed, otherwise the function supplies “false”.

- The name of characteristic is optional. If no characteristic name is indicated, the current characteristic will be used.
- The alignment is optional. If no alignment is specified, the base alignment will be used.
- `writeActualsToVDA(["characteristic"],"file_name")` exports stylus center point coordinates.

- `writeActualsToVDA(["characteristic",][true|false,]"file_name")` exports surface points in case of “true” and, in case of “false”, stylus center point coordinates.

Examples:

```
writeActualsToVDA("diameter1",true,"c:\test.vda")
```

exports surface points.

```
writeActualsToVDA("diameter1","c:\test.vda")
```

exports stylus center point coordinates.

```
writeActualsToVDA("diameter1",false,"c:\test.vda")
```

exports stylus center point coordinates.

writeBliskFinishFile

Writes the bliskfinish.xml file for communication with BLADE PRO. The syntax is:

```
writeBliskFinishFile("blisk name")
```

“blisk name” can be any name. Therefore, the “blisk name” must be used in the corresponding `writeBliskStartFile` command.

You use this command in the postsettings if you defined a “Turbine Blade” element in a loop.

writeBliskStartFile

Writes the bliskstartup.xml file for communication with BLADE PRO. The syntax is:

```
writeBliskStartFile("blisk name")
```

“blisk name” can be any name. Therefore, the “blisk name” must be used in the corresponding `writeBliskFinishFile` command.

You use this command in the presettings if you defined a “Turbine Blade” element in a loop.

writeCurveDistanceAsPlaneToFile

Writes curve points and the associated curve distances from a “curve distance” characteristic in the form of coordinates of a measured virtual plane to a file. The syntax is:

```
writeCurveDistanceAsPlaneToFile("characteristic",axis number,"file name")
```

The axis number may be 1, 2, or 3, corresponding then to X, Y, or Z. The curve distances pertaining to the nominal points of the inner curve (calculation method as defined in the characteristic) are used as coordinates in the specified axis direction; the other two coordinates of the plane points to be generated are filled with the corresponding coordinates of the inner curve's nominal points.

Example:

```
writeCurveDistanceAsPlaneToFile("Curve distance1",1,"c:\A\testfile.txt")
```

The Y and Z values of the nominal points are complemented in the X coordinate by the corresponding curve distances and written to the `c:\A\testfile.txt` file.

writeDiffCoordSysToFile

Calculates the difference matrix of two coordinate systems and outputs it as a file. The difference matrix of the two coordinate systems has the following structure:

$$\begin{matrix} a_{11} & a_{12} & a_{13} & x \\ a_{21} & a_{22} & a_{23} & y \\ a_{31} & a_{32} & a_{33} & z \\ 0 & 0 & 0 & 1 \end{matrix}$$

The output file is composed of the a_{nm} components of the matrix, appearing line-by-line, and of the X, Y and Z vector components. A blank is used as a separator. Thus, the file contains a line with the following structure:

```
a11 a12 a13 a21 a22 a23 a31 a32 a33 x y z
```

Each one of the twelve numbers is output with 14 decimal places.

The syntax is:

```
writeDiffCoordSysToFile([Name1,[Index1,][Actual1,]] Name2,[Index2,][Actual2,]file_name)
```

Meaning of the parameters:

Parameter	Meaning	
Name1	Name of the first coordinate system	Optional. If no name is specified, the base alignment will be used. In this case, Index1 and Actual1 are not needed either.
Index1	Loop index of the first coordinate system	optional if Name1 is specified. Default value: the current loop index.

Parameter	Meaning	
Actual	true: actual coordinate system false: nominal coordinate system	Optional. Default value: “true”.
Name2	Name of the second coordinate system	Name2 is mandatory.
Index2	Loop index of the second coordinate system	Optional. Default value: the current loop index.
Actual2	true: actual coordinate system false: nominal coordinate system	Optional. Default value: “true”.
File name	Path and file name of the output file	Absolute or relative specification possible. With a relative path specification, the actual value directory of the measurement plan is used as the basis.

writePointsCloudToFile

Writes the points of a point cloud into a session file. The syntax is:

```
writePointsCloudToFile([point_cloud_name,]file_name)
```

“point cloud name” is the name of a point set or a graphic point set.

If no parameter is indicated, the current feature will be used. This, however, is only practical if the command is located in the postparameters of a point cloud or a graphic point cloud.

writeUserAttributes

Exports the characteristic attributes with values into an ASCII file. The syntax is:

```
writeUserAttributes([[path_name\]file_name],[attribut1[,attribut2[,attribut3...]]])
```

`writeUserAttributes()` exports all characteristic attributes into the *userattributes.txt* file in the measurement plan directory.

`writeUserAttributes(file_name,[attribut1[,attribut2[,attribut3...]]])` exports the specified attributes into the file in the measurement plan directory.

The file is a table file. It contains lines and columns separated by tabulator.

- The first line contains the column headings (1st column: identifier, 2nd to nth column: keyword of the respective characteristic attribute).
- Any further line contains the characteristic name in the first column and the values of the corresponding attributes in the other columns.

PCM functions: Measurement-specific and measurement plan-specific functions

NOTE

Coordinate values returned by getActual, getNominal and measure always refer to the base alignment of the current feature.

addCF

Adds additional characteristics to the scope of measurement. The syntax is:

```
addCF(CharacteristicName1,CharacteristicName2, …)
```

Or

```
addCF(ListName)
```

You can specify individual characteristics or groups as `characteristic name`.

NOTE

Characteristics added in this way are *always* checked even if they are not selected at CNC start.

addME

Adds additional features to the scope of measurement. The syntax is:

```
addME(feature_name1,feature_name2, …)
```

NOTE

Only effective, if “From Feature List” is selected under **Order of run**.

baseSystem

Returns characteristics of the base alignment. The syntax is:

```
baseSystem().characteristic
```

Possible values of “characteristic” are: “x”, “y”, “z”, “valueA”, “euler1”, “euler2”, and “euler3”:

Function	Return value
baseSystem().x	X value of base alignment
baseSystem().y	Y value of base alignment
baseSystem().z	Z value of base alignment
baseSystem().valueA	valueA value of base alignment
baseSystem().euler1	euler1 of base alignment in rads (angle of inclination)
baseSystem().euler2	euler2 of base alignment in rads (angle of inclination)
baseSystem().euler3	euler3 of base alignment in rads (angle of inclination)

The formula for calculating the plane angle of the base alignment is:

```
plane angle = (euler1 - euler3) * 180  / π
```

Example: `result=baseSystem().x`

The X value of the base alignment is written into the “result” variable.

clearCAD

Deletes the contents of the CAD window. The syntax is:

```
clearCAD()
```

clearParameter

Deletes all or individual variables. The syntax is:

```
clearParameter([parametername1,parametername2, ...])
```

Examples:

clearParameter("Value_A")	deletes the variable Value_A
clearParameter("Value_A","Value_B")	deletes the variables Value_A and Value_B
clearParameter()	deletes all variables

The clearParameter function guarantees that no old undesired values are assigned to variables. If the formula accesses a deleted variable, CALYPSO issues the same warning as for a variable that is not assigned.

compute

Computes the value of a term or a single PCM parameter indicated as character string. The syntax is:

```
compute(CharacterString)
```

Example:

Given:

- A_Hugo=4711
- Part1="A"
- Part2="Hugo"
- Compl_para=Part1+Part2

Then “compute(Compl_para)” returns the value 4711.

defineRecheckPartNumber

Defines the special part number for the remeasurement (variable “part number”) The syntax is:

```
defineRecheckPartNumber(PCM expression)
```

By default, the “Incremental part Number” is the identification for the unique assignment of the data in the remeasurement. If this default is to be deviated from, you can define you own identifier with `defineRecheckPartNumber`.

The function can thus be used for the definition of a distinct “badElementFormula” function for the control of remeasurements.

Example:

```
// Define recheck-partnumber by PCM-term
RECHECK_PARTNB_TERM="getRunID()+getRecordHead("order")"
defineRecheckPartNumber(RECHECK_PARTNB_TERM)
```

defineFunctionEnd

Marks the end of the function definition. The syntax is:

```
defineFunctionEnd(ReturnValue)
```

defineFunctionStart

Marks the beginning of the function definition. The syntax is:

```
defineFunctionStart(FunctionName)
```

differenceSystem

Returns characteristics of an iterative alignment. The syntax is:

```
differenceSystem().characteristic
```

Possible values of “characteristic” are: “x”, “y”, “z”, “valueA”, “euler1”, “euler2”, and “euler3”:

Function	Return value
differenceSystem().x	Difference of the last two x values
differenceSystem().y	Difference of the last two y values
differenceSystem().z	Difference of the last two z values
differenceSystem().valueA	Difference of the last two valueA values
differenceSystem().euler1	Difference of the last two euler1 in rads (angle of inclination)
differenceSystem().euler2	Difference of the last two euler2 in rads (angle of inclination)
differenceSystem().euler3	Difference of the last two euler3 in rads (angle of inclination)

The formula for calculating the plane angle of the alignment is:

```
plane angle = (euler1 - euler3) * 180  / π
```

Example: `result=differenceSystem().x`

The difference of the x values of the last two iterations is written into the “result” variable.

endInspection()

Properly terminates the measurement plan. Contrary to canceling via `cncBreak()`, the current characteristic or feature will be processed, the measurement plan will be terminated and the intended printouts will be generated. The syntax is:

```
endInspection()
```

Example: `endInspection()`

The measurement plan run is concluded.

executeFunctionNamed

Executes a function that has been previously defined between `defineFunctionStart` and `defineFunctionEnd`. The syntax is:

```
executeFunctionNamed(function name)
```

The function must have been previously defined between `defineFunctionStart(function name)` and `defineFunctionEnd(return value)`.

Example:

```
defineFunctionStart("SUM")
C=A+B
defineFunctionEnd("C")
```

```
A=13
B=5
ERG=executeFunctionNamed("SUM")
```

The ERG value is here 18.

getActual

Used *without* an argument, it returns the current value of the characteristic. The syntax is:

```
getActual()
```

In conjunction with an argument, it returns a certain current value of a feature, coordinate system or bore pattern.

The syntax is:

```
getActual(feature_name[,[loop_index][,step_number]]).attribute
```

The “feature name” is a character string and can be indexed directly or with a variable (for example, a loop variable):

```
getActual("Cone",3).x
```

Or

```
getActual("Cone",LOOP3).x
```

The “Step Number” is only used for the stepped cylinder:

```
getActual("StepCylinder1",,4).diameter
```

“attribute” is a dummy for the following:

NOTE
Only matching attributes can be used for the respective features.

Attribute	Return value
a1	Angle One
a2	Angle Two
angleForDisplay	Angle
angleSegment	Angle range
apexAngle	Cone angle
apexAngleHalf	Half cone angle
boreIsMissing	“true” in case of missing bore, “false” if a bore exists (see Recognizing a missing bore in the operating instructions for the base module)
coordPolAngle	Polar coordinate, angle

Attribute	Return value
coordPolHeight	Polar coordinate, height
coordPolRadius	Polar coordinate, radius
Deviation	Deviation in the set tolerance mode
diameter	Diameter
diameter2	Diameter two of the ellipse
Distance	Distance of a symmetry point
elementIsMissing	“true” if it has not been possible to measure the feature and “false” if it has been measured (no probing) (see Continue at missing probing in the operating instructions for the base module)
endAngleUAxis	Angle of the second touch point (circle in contour best fit)
endPointA	Angle of the end point in polar coordinates
endPointH	Height in the z axis of the end point in polar coordinates
endPointR	Radius of the end point in polar coordinates
endPointX	x value of the end point of a 2D line
endPointY	y value of the end point of a 2D line
endPointZ	z value of the end point of a 2D line
expansionA	Length (rectangle or slot)
expansionB	Width (rectangle or slot)
focus1A	Angle of the first focal point in polar coordinates
focus1H	Height in the z axis of the first focal point in polar coordinates
focus1R	Radius of the first focal point in polar coordinates
focus1X	x value of the first focal point of an ellipse
focus1Y	y value of the first focal point of an ellipse
focus1Z	z value of the first focal point of an ellipse
focus2A	Angle of the second focal point in polar coordinates
focus2H	Height in the z axis of the second focal point in polar coordinates
focus2R	Radius of the second focal point in polar coordinates
focus2X	x value of the second focal point of an ellipse
focus2Y	y value of the second focal point of an ellipse
focus2Z	z value of the second focal point of an ellipse
form	Form Error
gap	Gap thickness between contour and fitted circle
getCamLift	Maximum curve lift or cam lift

Attribute	Return value
getCamLiftAngle	Angle (polar location) of the maximum curve lift or cam lift in the curve's polar coordinate system
getCamLiftAngleAccPnt	Angle (polar location) of the maximum cam lift. The first nominal point in ascending direction is used as the datum (angle = 0). The queried angle corresponds to the angle written into a text file when the results are output.
getCamLiftIndex	Numer of the nominal point with the maximum curve lift
getOffsetKind	Tolerance offset of the shape of zone
getOffsetValue	Shape of zone of the curve form
inclinationAngle	Tilt angle
len	Length
length	Length of a curve
materialDetected	"true", if material present; "false", if material absent; "9999" in case of error (see Detecting material in the operating instructions for the basic module)
maxDev	Maximum error
midPoint	Center point of a circle / 2D line
minDev	Minimum deviation
radius	Radius
radiusD2	Radius 2
rotationAngle	Angle of rotation
rotX	Rotation around the X axis
rotY	Rotation around the Y axis
rotZ	Rotation around the Z axis
sigma	Standard deviation
transX	Translation in X direction
transY	Translation in Y direction
transZ	Translation in Z direction
x	X value of the reference point X value of the origin for coordinate systems and bore patterns
xMaxDev	Maximum error in the X direction
xMinDev	Minimum error in the X direction
y	Y value of the reference point Y value of the origin for coordinate systems and bore patterns

Attribute	Return value
yMaxDev	Maximum error in the Y direction
yMinDev	Minimum error in the Y direction
z	Z value of the reference point Z value of the origin for coordinate systems and bore patterns
zMaxDev	Maximum error in the Z direction
zMinDev	Minimum error in the Z direction

For the Surface measurement with ROTOS feature, the following attributes can be used:

Attribute	Return value
Pt	Profile depth
Pz	Averaged profile depth
Ra	Arithmetic mean roughness
RdeltaC	Profile plateau ratio of the R profile
Rmr	Material ratio of the profile
Rp	Average smoothing depth
Rq	Roughness mean square
Rt	Maximum roughness depth
Rv	Average groove depth
Rz	Average roughness depth
Wa	Arithmetic mean waviness
Wmr	Profile material ratio on W profile
WmrC	Profile material ratio on W profile, basic abrasion
Wt	Peak to valley
Wz	Averaged peak to valley

For the qualification feature of ROTOS, the following attributes can be used:

Attribute	Return value
average	Average percentage deviation from existing calibrated value
meas1 ... meas5	Average percentage deviation of the nth measurement from existing calibrated value

getCFAttribute

Returns the current value of an attribute. Without definition of the characteristic name only usable within functions. The syntax is:

```
getCFAttribute(AttributeName)
```

Or

```
getCFAttribute(CharacteristicName,AttributeName)
```

getCFGroupname

Returns the currently highest group name to the characteristic. Without definition of the characteristic name only usable within functions. The syntax is:

```
getCFGroupname(characteristic name)
```

Or

```
getCFGroupname()
```

getCFGroupname2

Returns the currently lowest group name to the characteristic. Without definition of the characteristic name only usable within functions. The syntax is:

```
getCFGroupname2(characteristic name)
```

Or

```
getCFGroupname2()
```

getCornerPointFromCAD

Returns one of two extreme corner points of the CAD model in the base alignment. The desired corner point must be defined. The syntax is:

```
getCornerPointFromCAD("extremum")
```

“extremum” is a dummy for the following:

extremum	Result
min	Vector of the XYZ minimum
max	Vector of the XYZ maximum

getFunctionObjectName

Returns the name of the current characteristic. The syntax is:

```
getFunctionObjectName()
```

Can be used in own programmed PCM functions to enable access to the current characteristic. An example is the programmed "badElementFormula" function used for the definition of a remeasurement criterion.

getMaxActual

Returns the value of the highest error of all position evaluations for a best fit of bore pattern and the calculated maximum for the 2 point diameter or the corresponding angle. The syntax is:

– Best fit of bore pattern:

```
getMaxActual("name of best fit").deviation
```

– 2 point diameter, maximum:

```
getMaxActual("feature_name[,loop_index]").actual
```

– 2 point diameter, maximum angle:

```
getMaxActual("feature_name[,loop_index]").angleOfRadiusPoint
```

Enter the name of the characteristic without the "^Max" suffix.

getMinActual

Returns the calculated minimum for a 2 point diameter or the corresponding angle. The syntax is:

– 2 point diameter, minimum:

```
getMinActual("feature_name[,loop_index]").actual
```

– 2 point diameter, minimum angle:

```
getMinActual("feature_name[,loop_index]").angleOfRadiusPoint
```

Enter the name of the characteristic without the "^Min" suffix.

getNominal

Returns a certain nominal of a feature. The syntax is:

```
getNominal("feature_name").characteristic
```

See the table above for the possible values of "characteristic".

getParameterNamed

Returns the value of the parameter whose name is the value of the `parametername` character string variable. The syntax is:

```
getParameterNamed(ParameterName)
```

To ensure access to parameter values, you can only use fixed parameter names in expressions. This function allows you to define a variable parameter name.

Example 1:

```
diameter7 = 34.12

i = 7

ParameterName = "diameter" + text(i)

Length = getParameterNamed(ParameterName)
```

Then the “Length” variable has the current value 34.12.

Example 2:

```
color7="yellow"

i=7

CurColor="color" + text(i)

Basic color = getParameterNamed(CurColor)
```

Now, the “Basic color” variable has the value of the `"yellow"` character string.

getRecheckMode

Returns the current mode for repeat measurements. The syntax is:

```
getRecheckMode()
```

The return value is 1 if repeat measurements are activated and it is 0 if repeat measurements are deactivated.

getRESULTS

Returns the directory in which the results of the measurement plan are stored. The syntax is:

```
 getRESULTS()
```

getRunID

Returns the name of the mini-plan selected for the CNC start. The syntax is:

```
getRunID()
```

getStartSetting

Returns CNC start parameter The syntax is:

```
getStartSetting("parameter")
```

Parameters	Dialog element	Possible return values:
`selection`	Selection	– `"complete"` – `"actual"` – Name of characteristics group
`clearOldRes`	Clear existing results	– true – false
`meMode`	Order of the run	– `"cf"` – `"me"`
`naviMode`	Navigate feature to feature	– `"byUser"` – `"automatic"`
`runMode`	Run Mode	– `"normal"` – `"slowToFirst"` – `"stepMode"` – `"manually"`
`baseSystemType`	Type of alignment	– `"baseSystem"` – `"startSystem"`
`baseSystemRealName`	Name of alignment	Name of alignment
`printer`	Printer	– `TRUE` – `False`
`speed`	Speed	– Value as string – `"VMax"`
`manAlignment`	Manual Alignment	– `TRUE` – `False`
`activeTechnology`	Strategy to be performed	Name of strategy

inspectionToleranceState

Delivers the tolerance status of the measurement plan, as far as it has been defined and returns the number of features with this tolerance status.

It makes sense to call the function in the postsettings of the measurement plan to query the status.

The syntax is:

```
inspectionToleranceState("ToleranceStatus")
```

The `ToleranceStatus` parameter can adopt the following values:

Tolerance status	Return value
`total`	Total number of characteristics
`inTolerance`	Number of characteristics within tolerance
`outOfLimit`	Number of characteristics exceeding the warning limit
`outOfTolerance`	Number of characteristics out of tolerance
`noResult`	Number of non-calculated characteristics
`wpSystem`	Number of coordinate systems
`wpSystemNoResult`	Number of non-calculated coordinate systems

If no parameter is specified, the measurement plan status will be returned:

Return value	Meaning
#notDefined	Status has not been defined yet.
#inTolerance	Measurement plan within tolerance: All features are within the specified tolerances.
#outOfLimit	Measurement plan outside the limits: At least one feature of the measurement plan is outside the specified warning limits.
#outOfTolerance	Measurement plan out of tolerance: At least one feature of the measurement plan is outside the specified tolerances.

Example:

```
inspectionToleranceState()
```

Supplies the tolerance status of the measurement plan.

isParameterDefined

Return as the logical value whether or not a variable has been defined. The syntax is:

```
isParameterDefined(ParameterName[,loop index})
```

The return value is `true` if the variable or the component of a list designated by the loop index has been defined. Otherwise, the return value is `false`.

loadCADFile

Loads a file of the **.sab* or **.sat* type.

The syntax is:

```
 loadCADFile(file name)
```

The complete file path is given as “file name”.

Example:

```
loadCADFile(getActualInspectionDir()+"\model.sab")
```

Measure

In conjunction with an argument, it returns a certain current value of a feature, coordinate system or bore pattern.

The syntax is:

```
measure("FeatureName"[,loop index]).characteristic
```

The “feature name” can be indexed directly or with a variable (for example, a loop variable):

```
measure("Cone",3).x
```

Or

```
measure("Cone",LOOP3).x
```

See the table above for the possible values of “characteristic”.

Used *without* an argument, it returns the current value of the characteristic. The syntax is:

```
measure()
```

measuringForce

Allows the measuring force in a feature or in the measurement plan to be set or read. The unit for the measuring force is mN.

The syntax is:

```
measuringForce([measuring_force_in_mN,]["feature"])
```

- If the function is called without parameters, the measuring force of the current feature will be returned (“current feature” stands here for the Feature if the function is called in the presettings or postsettings of a feature; otherwise it stands for the Measurement Plan).
- When the name of a feature is given as parameter, the measuring force of this feature will be returned.
- If the function is called in the presettings or postsettings of the measurement plan and if the measuring force is specified as parameter, the measuring force will be set for the entire measurement plan. If the function is called with a measuring force in the presettings and postsettings of a feature, the measuring force of the feature will be set.

– If the measuring force and the name of a feature have been specified as parameters, the measuring force of the feature will be set directly.

Examples:

`myForce=measuringForce()`

Returns the measuring force of the current feature or of the measurement plan and saves the value in the myForce variable.

`myForce=measuringForce("Cylinder1")`

Returns the measuring force of the “Cylinder1” feature and saves the value in the myForce variable.

`measuringForce(100)`

Sets the measuring force of the current feature or of the measurement plan to 100 mN.

`measuringForce(100,"Cylinder1")`

Sets the measuring force of the “Cylinder1” feature to 100 mN.

numberOfPointDistanceExceed

Specifies the number of curve points with a deviation greater than the reference value.

The syntax is:

`numberOfPointDistanceExceed("CurveFormName", reference value)`

The function returns the number of points.

Example:

`numberOfPointDistanceExceed ("CurveForm1", 0.025)` counts the number of curve points with a deviation > 0.0025 and returns the number.

numberOfPointDistanceLessThan

Specifies the number of curve points with a deviation smaller than the reference value.

The syntax is:

```
numberOfPointDistanceLessThan("KurvenFormName", reference
value)
```

The function returns the number of points.

Example:

`numberOfPointDistanceLessThan ("CurveForm2", 0.001)` counts the number of curve points with a deviation < 0.001 and returns the number.

outputMultiQdas

Creates a separate Q-DAS evaluation according to a predefinable characteristic attribute. The syntax is:

```
outputMultiQdas(attribute name)
```

setCF

Defines the characteristics belonging to the scope of measurement. The syntax is:

```
setCF(CharacteristicName1,CharacteristicName2, …)
```

Or

```
setCF(list name)
```

You can specify individual characteristics or groups as `characteristic name`.

NOTE

The characteristics defined in this way replace all previously selected characteristics.

setCFAttribute

Sets a characteristic attribute on a certain value for the current characteristic. The syntax is:

```
setCFAttribute (keyword, value)
```

The characteristic attributes are written as an additional column in the table file for characteristics (*chr.txt).

Example:

The output settings of the “Diam_4711” characteristic contain a value assignment for ParamX as well as setCFAttribute:

```
setCFAttribute ("u_resultX",ParamX)
```

Result: An additional column labeled with “u_resultX” is created in the table file. The line of the “Diam_4711” characteristic in this column contains the assigned value of ParamX.

setElementStrategy

Defines the current strategy. The syntax is:

```
°setElementStrategy([feature_name],strategy_name)
```

- In the input parameters of a feature, this command has effect on the current feature.
- In the input parameters of the measurement plan, this command has effect on the specified feature.

setFilter

Defines the filter. The syntax is:

```
setFilter(measurement_plan_element_name,subelement,filter_method,filter_type,filter_criteria,connect_segments)
```

“measurement_plan_element_name” is the name of the feature, the characteristic, or the alignment.

If the first parameter is not the name of a feature, then the second parameter should define the desired subelement.

“subelement” defines the desired subelement if `measurement_plan_element_name` does not represent a feature.

Value	Meaning
(Undefined)	Feature or feature 1
`"featureElement"`	Feature
`"featureElement1"`	Feature 1
`"featureElement2"`	Feature 2
`"primaryDatum"`	Primary Datum
`"secondaryDatum"`	Secondary Datum
`"tertiaryDatum"`	Tertiary Datum
`"rotationSpace"`	Spatial Rotation
`"rotationPlane"`	Planar Rotation
`"translationX"`	X Origin
`"translationY"`	Y Origin
`"translationZ"`	Z Origin

“filter_method” defines the filter method.

Value	Meaning
`"off"`	Filter off
`"fromFeature"`	Adoption from feature (only for characteristics)
`"gauss"`	Gaussian filter

Value	Meaning
`"spline"`	Spline filter
`"2 RC"`	2RC filter

„filter_type“ defines the type of filter.

Value	Meaning
`"low"`	Low-pass
`"band"`	Band-pass
`"high"`	High-pass

“filter_criteria” defines the filter criterion and is followed by one or two numerical values.

Value	Meaning
`"undulation"`	Undulations Per Revolution
`"wavelength"`	Wavelength Lc

“connect_segments” indicates whether the segments are connected.

Value	Meaning
True	Segments are connected
False	Segments are not connected

Return values

Return value	Meaning
0	Assignment successful
1	Invalid numbers of parameters
2	Elements not found
3	Filter (for the element) not permitted
4	Invalid data type
5	Invalid value

Examples

```
setFilter(feature_name,"off")
```

Disables the filter for the specified feature.

```
setFilter(feature_name,"gauss","low","wavelength",3.0)
```

Sets a filter for the specified feature with Gauss, Low-pass, Wavelength Lc 3.0 mm, and Connect Segments = off.

```
setFilter(feature_name,"2 RC","band","undulation",50,300)
```

Sets a filter for the specified feature with 2 RC, Band-pass, Undulations Per Revolution from 50 W/U to 300 W/U, and Connect Segments = off.

```
setFilter(characteristic_name,"featureElement","fromFeature")
```

Sets a filter for the specified characteristic, feature, applying the filter from the feature.

```
setFilter(characteristic_name,"primaryDatum","2
RC","band","undulation",50,300,true)
```

Sets a filter for the specified characteristic with Primary Datum, 2 RC, Band-pass, Undulations Per Revolution from 50 W/U to 300 W/U, and Connect Segments = on.

setGeometryInnerOuter

Defines the feature as an inside or outside feature. The syntax is:

```
setGeometryInnerOuter("Feature ID")
```

This means:

Feature ID	Effect
OUTER	Set to outside feature
INNER	Set to inside feature

This command only acts on the feature in whose input parameters it is contained.

NOTE

A text parameter enables variable specification of feature IDs.

setInspectionStrategy

Defines the current strategy for the entire measurement plan. The syntax is:

```
setInspectionStrategy(StrategyName)
```

This command must be contained in the input parameters of the measurement plan. A blank strategy name will cause selection of the evasion strategy.

setMissingBore

Defines a search distance and reduces the travel speed until reaching the search distance before the first probing. The syntax is:

```
setMissingBore ([feature_name,]search_distance)
```

The feature name can be omitted in the pre-parameters of the current feature.

You will be informed about a missing bore via

```
getActual("feature_name").boreIsMissing
```

The return value is `true` or `false`.

setOutlier

Defines the outlier elimination. The syntax is:

```
setOutlier(measurement_plan_element_name,subelement,factor
i.P.,factor o.P,Area data red.,iterations,filter_criteria)
```

“measurement_plan_element_name” is the name of the feature, the characteristic, or the alignment.

If the first parameter is not the name of a feature, then the second parameter should define the desired subelement.

Value	Meaning
(Undefined)	Feature or feature 1
`"featureElement"`	Feature
`"featureElement1"`	Feature 1
`"featureElement2"`	Feature 2
`"primaryDatum"`	Primary Datum
`"secondaryDatum"`	Secondary Datum
`"tertiaryDatum"`	Tertiary Datum
`"rotationSpace"`	Spatial Rotation
`"rotationPlane"`	Planar Rotation
`"translationX"`	X Origin
`"translationY"`	Y Origin
`"translationZ"`	Z Origin

“factor i.P” defines the factor inside the workpiece.

Value	Meaning
Number	Factor for outliers inside the workpiece
`"off"`	Outlier elimination off
`"fromFeature"`	Adoption from feature (only for characteristics)

“factor o.P” defines the factor outside the workpiece.

Value	Meaning
Number	Factor for outliers outside the workpiece
`"off"`	Outlier elimination off
`"fromFeature"`	Adoption from feature (only for characteristics)

“Area data red.” defines the range of data reduction.

Value	Meaning
`"onlyOutliers"`	Only Outliers
`"adjacentPoints"`	Adjacent points i. Number ii. To Computed Feature
`"fromFeature"`	Adoption from feature (only for characteristics)

“iterations” defines the number of iterations.

“filter criteria” followed be two numerical values defines the filter criterion.

Value	Meaning
`"undulation"`	Undulations Per Revolution
`"wavelength"`	Wavelength Lc

Return values

Return value	Meaning
0	Assignment successful
1	Invalid numbers of parameters
2	Elements not found
3	Outlier elimination (for the element) not permitted
4	Invalid data type
5	Invalid value

Examples

```
setOutlier(feature_name,"off")
```

Disables the outlier elimination for the given feature.

```
setOutlier(feature_name,3,3,"onlyOutliers",1,"wavelength",0,1000)
```

Sets the outlier elimination for the given feature with factors for outliers 3 / 3, Only Outliers, Number of iterations 1, Wavelength Lc from 0 mm to 1000 mm.

```
setOutlier(feature_name,5,5,"adjacentPoints",0,false,1,"undulation",0,4000)
```

Sets the outlier elimination for the given feature with factors for outliers 5 / 5, Adjacent points, Number 0, To Computed Feature OFF, Number of iterations 1, Undulations Per Revolution from 0 W/U to 4000 W/U.

```
setOutlier(characteristic_name,"featureElement","fromFeature")
```

Sets the outlier elimination for the given characteristic, feature, applying the filter from the feature.

```
setOutlier(characteristic_name,"primaryDatum",5,5,"adjacentPoints",0,true,2,"undulation",0,4000)
```

Sets the outlier elimination for the given characteristic, Primary Datum, with factors for outliers 5 / 5, Adjacent points, Number 0, To Computed Feature ON, Number of iterations 1, Undulations Per Revolution from 0 W/U to 4000 W/U.

setParameterNamed

Assigns a value to a variable during runtime. The syntax is:

```
setParameterNamed(parameter_name,value[,loop index})
```

The parameters have the following meaning:

Parameters	Meaning
Parameter name	Variable with the name of the variable or the variable's name itself
Value	Variable with the value to be assigned or the value itself
Loop index	Variable for the index in the list or the index itself

Example: If the value "Delivery address" with the string `"Main plant Untersiemau"` has been assigned to the Address type variable during runtime, then this results from the command `setParameterNamed(Adress type,Adresse)` assigning the string `"Main plant Untersiemau"` to the `Delivery address` variable.

setRecheckMode

Sets the mode for repeat measurement for the measurement plan. The syntax is:

```
setRecheckMode(Recheck[,Reset][,Maximum])
```

The parameters have the following meaning:

Parameters	Meaning
Recheck = 0	Repeat measurement deactivated
Recheck = 1	Repeat measurement activated
Reset = 0	No reset of part numbers
Reset = 1	All part numbers are reset
Maximum	The maximum number of repeat measurements. 0 = no limitation

setRunID

Sets the mini-plans selected for the CNC start. The syntax is:

```
°setRunID(characteristic group name1,characteristic group
name2, …)
```

The previously saved selection is overwritten. When defining several mini-plans in a call, the total selection forms the scope of measurement.

setShaded

Renders the displayed CAD model. The syntax is:

```
setShaded()
```

setWireframe

Shows the displayed CAD model as a wire frame model. The syntax is:

```
setWireframe()
```

PCM functions: Measurement-specific and measurement plan-specific functions

NOTE

Coordinate values returned by getActual, getNominal and measure always refer to the base alignment of the current feature.

addCF

Adds additional characteristics to the scope of measurement. The syntax is:

```
addCF(CharacteristicName1,CharacteristicName2, …)
```

Or

```
addCF(ListName)
```

You can specify individual characteristics or groups as `characteristic name`.

NOTE
Characteristics added in this way are *always* checked even if they are not selected at CNC start.

addME

Adds additional features to the scope of measurement. The syntax is:

```
addME(feature_name1,feature_name2, …)
```

NOTE
Only effective, if "From Feature List" is selected under **Order of run**.

baseSystem

Returns characteristics of the base alignment. The syntax is:

```
baseSystem().characteristic
```

Possible values of "characteristic" are: "x", "y", "z", "valueA", "euler1", "euler2", and "euler3":

Function	Return value
baseSystem().x	X value of base alignment
baseSystem().y	Y value of base alignment
baseSystem().z	Z value of base alignment
baseSystem().valueA	valueA value of base alignment
baseSystem().euler1	euler1 of base alignment in rads (angle of inclination)
baseSystem().euler2	euler2 of base alignment in rads (angle of inclination)
baseSystem().euler3	euler3 of base alignment in rads (angle of inclination)

The formula for calculating the plane angle of the base alignment is:

```
plane angle = (euler1 - euler3) * 180  / π
```

Example: `result=baseSystem().x`

The X value of the base alignment is written into the "result" variable.

clearCAD

Deletes the contents of the CAD window. The syntax is:

```
clearCAD()
```

clearParameter

Deletes all or individual variables. The syntax is:

```
°clearParameter([parameter_name1,parameter_name2, ...])
```

Examples:

clearParameter("Value_A")	deletes the variable Value_A
clearParameter("Value_A","Value_B")	deletes the variables Value_A and Value_B
clearParameter()	deletes all variables

The clearParameter function guarantees that no old undesired values are assigned to variables. If the formula accesses a deleted variable, CALYPSO issues the same warning as for a variable that is not assigned.

compute

Computes the value of a term or a single PCM parameter indicated as character string. The syntax is:

```
°compute(CharacterString)
```

Example:

Given:

- A_Hugo=4711
- Part1="A"
- Part2="Hugo"
- Compl_para=Part1+Part2

Then “compute(Compl_para)” returns the value 4711.

defineRecheckPartNumber

Defines the special part number for the remeasurement (variable “part number”) The syntax is:

```
defineRecheckPartNumber(PCM expression)
```

By default, the “Incremental part Number” is the identification for the unique assignment of the data in the remeasurement. If this default is to be deviated from, you can define you own identifier with `defineRecheckPartNumber`.

The function can thus be used for the definition of a distinct “badElementFormula” function for the control of remeasurements.

Example:

```
// Define recheck-partnumber by PCM-term
RECHECK_PARTNB_TERM="getRunID()+getRecordHead("order")"
defineRecheckPartNumber(RECHECK_PARTNB_TERM)
```

defineFunctionEnd

Marks the end of the function definition. The syntax is:

```
defineFunctionEnd(ReturnValue)
```

defineFunctionStart

Marks the beginning of the function definition. The syntax is:

```
defineFunctionStart(FunctionName)
```

differenceSystem

Returns characteristics of an iterative alignment. The syntax is:

```
differenceSystem().characteristic
```

Possible values of “characteristic” are: “x”, “y”, “z”, “valueA”, “euler1”, “euler2”, and “euler3”:

Function	Return value
differenceSystem().x	Difference of the last two x values
differenceSystem().y	Difference of the last two y values
differenceSystem().z	Difference of the last two z values
differenceSystem().valueA	Difference of the last two valueA values
differenceSystem().euler1	Difference of the last two euler1 in rads (angle of inclination)
differenceSystem().euler2	Difference of the last two euler2 in rads (angle of inclination)
differenceSystem().euler3	Difference of the last two euler3 in rads (angle of inclination)

The formula for calculating the plane angle of the alignment is:

```
plane angle = (euler1 - euler3) * 180  / π
```

Example: `result=differenceSystem().x`

The difference of the x values of the last two iterations is written into the “result” variable.

endInspection()

Properly terminates the measurement plan. Contrary to canceling via `cncBreak()`, the current characteristic or feature will be processed, the measurement plan will be terminated and the intended printouts will be generated. The syntax is:

```
endInspection()
```

Example: `endInspection()`

The measurement plan run is concluded.

executeFunctionNamed

Executes a function that has been previously defined between `defineFunctionStart` and `defineFunctionEnd`. The syntax is:

```
executeFunctionNamed(function name)
```

The function must have been previously defined between `defineFunctionStart(function name)` and `defineFunctionEnd(return value)`.

Example:

```
defineFunctionStart("SUM")
C=A+B
defineFunctionEnd("C")

A=13
B=5
ERG=executeFunctionNamed("SUM")
```

The ERG value is here 18.

getActual

Used *without* an argument, it returns the current value of the characteristic. The syntax is:

```
getActual()
```

In conjunction with an argument, it returns a certain current value of a feature, coordinate system or bore pattern.

The syntax is:

```
getActual(feature_name[,[loop_index][,step_number]]).attribute
```

The “feature name” is a character string and can be indexed directly or with a variable (for example, a loop variable):

```
getActual("Cone",3).x
```

Or

```
getActual("Cone",LOOP3).x
```

The “Step Number” is only used for the stepped cylinder:

```
getActual("StepCylinder1",,4).diameter
```

“attribute” is a dummy for the following:

NOTE

Only matching attributes can be used for the respective features.

Attribute	Return value
a1	Angle One
a2	Angle Two
angleForDisplay	Angle
angleSegment	Angle range
apexAngle	Cone angle
apexAngleHalf	Half cone angle
boreIsMissing	“true” in case of missing bore, “false” if a bore exists (see Recognizing a missing bore in the operating instructions for the base module)
coordPolAngle	Polar coordinate, angle
coordPolHeight	Polar coordinate, height
coordPolRadius	Polar coordinate, radius
Deviation	Deviation in the set tolerance mode
diameter	Diameter
diameter2	Diameter two of the ellipse
Distance	Distance of a symmetry point
elementIsMissing	“true” if it has not been possible to measure the feature and “false” if it has been measured (no probing) (see Continue at missing probing in the operating instructions for the base module)
endAngleUAxis	Angle of the second touch point (circle in contour best fit)
endPointA	Angle of the end point in polar coordinates
endPointH	Height in the z axis of the end point in polar coordinates
endPointR	Radius of the end point in polar coordinates
endPointX	x value of the end point of a 2D line
endPointY	y value of the end point of a 2D line
endPointZ	z value of the end point of a 2D line
expansionA	Length (rectangle or slot)

Attribute	Return value
expansionB	Width (rectangle or slot)
focus1A	Angle of the first focal point in polar coordinates
focus1H	Height in the z axis of the first focal point in polar coordinates
focus1R	Radius of the first focal point in polar coordinates
focus1X	x value of the first focal point of an ellipse
focus1Y	y value of the first focal point of an ellipse
focus1Z	z value of the first focal point of an ellipse
focus2A	Angle of the second focal point in polar coordinates
focus2H	Height in the z axis of the second focal point in polar coordinates
focus2R	Radius of the second focal point in polar coordinates
focus2X	x value of the second focal point of an ellipse
focus2Y	y value of the second focal point of an ellipse
focus2Z	z value of the second focal point of an ellipse
form	Form Error
gap	Gap thickness between contour and fitted circle
getCamLift	Maximum curve lift or cam lift
getCamLiftAngle	Angle (polar location) of the maximum curve lift or cam lift in the curve's polar coordinate system
getCamLiftAngleAccPnt	Angle (polar location) of the maximum cam lift. The first nominal point in ascending direction is used as the datum (angle = 0). The queried angle corresponds to the angle written into a text file when the results are output.
getCamLiftIndex	Numer of the nominal point with the maximum curve lift
getOffsetKind	Tolerance offset of the shape of zone
getOffsetValue	Shape of zone of the curve form
inclinationAngle	Tilt angle
len	Length
length	Length of a curve
maxDev	Maximum error
midPoint	Center point of a circle / 2D line
minDev	Minimum deviation
radius	Radius
radiusD2	Radius 2
rotationAngle	Angle of rotation

Attribute	Return value
rotX	Rotation around the X axis
rotY	Rotation around the Y axis
rotZ	Rotation around the Z axis
sigma	Standard deviation
transX	Translation in X direction
transY	Translation in Y direction
transZ	Translation in Z direction
x	X value of the reference point X value of the origin for coordinate systems and bore patterns
xMaxDev	Maximum error in the X direction
xMinDev	Minimum error in the X direction
y	Y value of the reference point Y value of the origin for coordinate systems and bore patterns
yMaxDev	Maximum error in the Y direction
yMinDev	Minimum error in the Y direction
z	Z value of the reference point Z value of the origin for coordinate systems and bore patterns
zMaxDev	Maximum error in the Z direction
zMinDev	Minimum error in the Z direction

For the Surface measurement with ROTOS feature, the following attributes can be used:

Attribute	Return value
Pt	Profile depth
Pz	Averaged profile depth
Ra	Arithmetic mean roughness
RdeltaC	Profile plateau ratio of the R profile
Rmr	Material ratio of the profile
Rp	Average smoothing depth
Rq	Roughness mean square
Rt	Maximum roughness depth
Rv	Average groove depth
Rz	Average roughness depth

Attribute	Return value
Wa	Arithmetic mean waviness
Wmr	Profile material ratio on W profile
WmrC	Profile material ratio on W profile, basic abrasion
Wt	Peak to valley
Wz	Averaged peak to valley

For the qualification feature of ROTOS, the following attributes can be used:

Attribute	Return value
average	Average percentage deviation from existing calibrated value
meas1 ... meas5	Average percentage deviation of the nth measurement from existing calibrated value

getCFAttribute

Returns the current value of an attribute. Without definition of the characteristic name only usable within functions. The syntax is:

```
getCFAttribute(AttributeName)
```

Or

```
getCFAttribute(CharacteristicName,AttributeName)
```

getCFGroupname

Returns the currently highest group name to the characteristic. Without definition of the characteristic name only usable within functions. The syntax is:

```
getCFGroupname(characteristic name)
```

Or

```
getCFGroupname()
```

getCFGroupname2

Returns the currently lowest group name to the characteristic. Without definition of the characteristic name only usable within functions. The syntax is:

```
getCFGroupname2(characteristic name)
```

Or

```
getCFGroupname2()
```

getCornerPointFromCAD

Returns one of two extreme corner points of the CAD model in the base alignment. The desired corner point must be defined. The syntax is:

```
getCornerPointFromCAD("extremum")
```

“extremum” is a dummy for the following:

extremum	Result
min	Vector of the XYZ minimum
max	Vector of the XYZ maximum

getFunctionObjectName

Returns the name of the current characteristic. The syntax is:

```
getFunctionObjectName()
```

Can be used in own programmed PCM functions to enable access to the current characteristic. An example is the programmed “badElementFormula” function used for the definition of a remeasurement criterion.

getMaxActual

Returns the value of the highest error of all position evaluations for a best fit of bore pattern and the calculated maximum for the 2 point diameter or the corresponding angle. The syntax is:

- Best fit of bore pattern:

```
getMaxActual("name of best fit").deviation
```

- 2 point diameter, maximum:

```
getMaxActual("feature_name[,loop_index]").actual
```

- 2 point diameter, maximum angle:

```
getMaxActual("feature_name[,loop_index]").angleOfRadiusPoint
```

Enter the name of the characteristic without the “^Max” suffix.

getMinActual

Returns the calculated minimum for a 2 point diameter or the corresponding angle. The syntax is:

- 2 point diameter, minimum:

```
getMinActual("feature_name[,loop_index]").actual
```

– 2 point diameter, minimum angle:

```
getMinActual("feature_name[,loop_index]").angleOfRadiusPoint
```

Enter the name of the characteristic without the “^Min” suffix.

getNominal

Returns a certain nominal of a feature. The syntax is:

```
getNominal("feature_name").characteristic
```

See the table above for the possible values of “characteristic”.

getParameterNamed

Returns the value of the parameter whose name is the value of the `parametername` character string variable. The syntax is:

```
getParameterNamed(ParameterName)
```

To ensure access to parameter values, you can only use fixed parameter names in expressions. This function allows you to define a variable parameter name.

Example 1:

```
diameter7 = 34.12

i = 7

ParameterName = "diameter" + text(i)

Length = getParameterNamed(ParameterName)
```

Then the “Length” variable has the current value 34.12.

Example 2:

```
color7="yellow"

i=7

CurColor="color" + text(i)

Basic color = getParameterNamed(CurColor)
```

Now, the “Basic color” variable has the value of the `"yellow"` character string.

getRecheckMode

Returns the current mode for repeat measurements. The syntax is:

```
getRecheckMode()
```

The return value is 1 if repeat measurements are activated and it is 0 if repeat measurements are deactivated.

getRESULTS

Returns the directory in which the results of the measurement plan are stored. The syntax is:

```
 getRESULTS()
```

getRunID

Returns the name of the mini-plan selected for the CNC start. The syntax is:

```
getRunID()
```

getStartSetting

Returns CNC start parameter The syntax is:

```
getStartSetting("parameter")
```

Parameters	Dialog element	Possible return values:
`selection`	Selection	– `"complete"` – `"actual"` – Name of characteristics group
`clearOldRes`	Clear existing results	– True – False
`meMode`	Order of the run	– `"cf"` – `"me"`
`naviMode`	Navigate feature to feature	– `"byUser"` – `"automatic"`
`runMode`	Run Mode	– `"normal"` – `"slowToFirst"` – `"stepMode"` – `"manually"`
`baseSystemType`	Type of alignment	– `"baseSystem"` – `"startSystem"`
`baseSystemRealName`	Name of alignment	Name of alignment
`printer`	Printer	– `TRUE` – `False`
`speed`	Speed	– Value as string – `"VMax"`

Parameters	Dialog element	Possible return values:
manAlignment	Manual Alignment	– TRUE – False
activeTechnology	Strategy to be performed	Name of strategy

inspectionToleranceState

Delivers the tolerance status of the measurement plan, as far as it has been defined and returns the number of features with this tolerance status.

It makes sense to call the function in the postsettings of the measurement plan to query the status.

The syntax is:

```
inspectionToleranceState("ToleranceStatus")
```

The `ToleranceStatus` parameter can adopt the following values:

Tolerance status	Return value
total	Total number of characteristics
inTolerance	Number of characteristics within tolerance
outOfLimit	Number of characteristics exceeding the warning limit
outOfTolerance	Number of characteristics out of tolerance
noResult	Number of non-calculated characteristics
wpSystem	Number of coordinate systems
wpSystemNoResult	Number of non-calculated coordinate systems

If no parameter is specified, the measurement plan status will be returned:

Return value	Meaning
#notDefined	Status has not been defined yet.
#inTolerance	Measurement plan within tolerance: All features are within the specified tolerances.
#outOfLimit	Measurement plan outside the limits: At least one feature of the measurement plan is outside the specified warning limits.
#outOfTolerance	Measurement plan out of tolerance: At least one feature of the measurement plan is outside the specified tolerances.

Example:

```
inspectionToleranceState()
```

Supplies the tolerance status of the measurement plan.

isParameterDefined

Return as the logical value whether or not a variable has been defined. The syntax is:

```
isParameterDefined(ParameterName[,loop index})
```

The return value is `true` if the variable or the component of a list designated by the loop index has been defined. Otherwise, the return value is `false`.

loadCADFile

Loads a file of the **.sab* or **.sat* type.

The syntax is:

```
 loadCADFile(file name)
```

The complete file path is given as “file name”.

Example:

```
 loadCADFile(getActualInspectionDir()+"\model.sab")
```

Measure

In conjunction with an argument, it returns a certain current value of a feature, coordinate system or bore pattern.

The syntax is:

```
 measure("FeatureName"[,loop index]).characteristic
```

The “feature name” can be indexed directly or with a variable (for example, a loop variable):

```
measure("Cone",3).x
```

Or

```
measure("Cone",LOOP3).x
```

See the table above for the possible values of “characteristic”.

Used *without* an argument, it returns the current value of the characteristic. The syntax is:

```
measure()
```

measuringForce

Allows the measuring force in a feature or in the measurement plan to be set or read. The unit for the measuring force is mN.

The syntax is:

```
measuringForce([measuring_force_in_mN,]["feature"])
```

- If the function is called without parameters, the measuring force of the current feature will be returned ("current feature" stands here for the Feature if the function is called in the presettings or postsettings of a feature; otherwise it stands for the Measurement Plan).
- When the name of a feature is given as parameter, the measuring force of this feature will be returned.
- If the function is called in the presettings or postsettings of the measurement plan and if the measuring force is specified as parameter, the measuring force will be set for the entire measurement plan. If the function is called with a measuring force in the presettings and postsettings of a feature, the measuring force of the feature will be set.
- If the measuring force and the name of a feature have been specified as parameters, the measuring force of the feature will be set directly.

Examples:

```
myForce=measuringForce()
```

Returns the measuring force of the current feature or of the measurement plan and saves the value in the myForce variable.

```
myForce=measuringForce("Cylinder1")
```

Returns the measuring force of the "Cylinder1" feature and saves the value in the myForce variable.

```
measuringForce(100)
```

Sets the measuring force of the current feature or of the measurement plan to 100 mN.

```
measuringForce(100,"Cylinder1")
```

Sets the measuring force of the "Cylinder1" feature to 100 mN.

numberOfPointDistanceExceed

Specifies the number of curve points with a deviation greater than the reference value.

The syntax is:

```
numberOfPointDistanceExceed("CurveFormName", reference value)
```

The function returns the number of points.

Example:

`numberOfPointDistanceExceed ("CurveForm1", 0.025)` counts the number of curve points with a deviation > 0.0025 and returns the number.

numberOfPointDistanceLessThan

Specifies the number of curve points with a deviation smaller than the reference value.

The syntax is:

```
numberOfPointDistanceLessThan("KurvenFormName", reference
value)
```

The function returns the number of points.

Example:

`numberOfPointDistanceLessThan ("CurveForm2", 0.001)` counts the number of curve points with a deviation < 0.001 and returns the number.

outputMultiQdas

Creates a separate Q-DAS evaluation according to a predefinable characteristic attribute. The syntax is:

```
outputMultiQdas(attribute name)
```

setCF

Defines the characteristics belonging to the scope of measurement. The syntax is:

```
setCF(CharacteristicName1,CharacteristicName2, …)
```

Or

```
setCF(list name)
```

You can specify individual characteristics or groups as `characteristic name`.

NOTE

The characteristics defined in this way replace all previously selected characteristics.

setCFAttribute

Sets a characteristic attribute on a certain value for the current characteristic. The syntax is:

```
setCFAttribute (keyword, value)
```

The characteristic attributes are written as an additional column in the table file for characteristics (*chr.txt).

Example:

The output settings of the “Diam_4711” characteristic contain a value assignment for ParamX as well as setCFAttribute:

```
setCFAttribute ("u_resultX",ParamX)
```

Result: An additional column labeled with “u_resultX” is created in the table file. The line of the “Diam_4711” characteristic in this column contains the assigned value of ParamX.

setElementStrategy

Defines the current strategy. The syntax is:

```
°setElementStrategy([feature_name],strategy_name)
```

– In the input parameters of a feature, this command has effect on the current feature.

– In the input parameters of the measurement plan, this command has effect on the specified feature.

setFilter

Defines the filter. The syntax is:

```
setFilter(measurement_plan_element_name,subelement,fil-
ter_method,filter_type,filter_criteria,connect_segments)
```

“measurement_plan_element_name” is the name of the feature, the characteristic, or the alignment.

If the first parameter is not the name of a feature, then the second parameter should define the desired subelement.

“subelement” defines the desired subelement if `measurement_plan_element_name` does not represent a feature.

Value	Meaning
(Undefined)	Feature or feature 1
`"featureElement"`	Feature
`"featureElement1"`	Feature 1
`"featureElement2"`	Feature 2
`"primaryDatum"`	Primary Datum
`"secondaryDatum"`	Secondary Datum
`"tertiaryDatum"`	Tertiary Datum
`"rotationSpace"`	Spatial Rotation
`"rotationPlane"`	Planar Rotation

Value	Meaning
`"translationX"`	X Origin
`"translationY"`	Y Origin
`"translationZ"`	Z Origin

“filter_method” defines the filter method.

Value	Meaning
`"off"`	Filter off
`"fromFeature"`	Adoption from feature (only for characteristics)
`"gauss"`	Gaussian filter
`"spline"`	Spline filter
`"2 RC"`	2RC filter

„filter_type“ defines the type of filter.

Value	Meaning
`"low"`	Low-pass
`"band"`	Band-pass
`"high"`	High-pass

“filter_criteria” defines the filter criterion and is followed by one or two numerical values.

Value	Meaning
`"undulation"`	Undulations Per Revolution
`"wavelength"`	Wavelength Lc

“connect_segments” indicates whether the segments are connected.

Value	Meaning
True	Segments are connected
False	Segments are not connected

Return values

Return value	Meaning
0	Assignment successful

Return value	Meaning
1	Invalid numbers of parameters
2	Elements not found
3	Filter (for the element) not permitted
4	Invalid data type
5	Invalid value

Examples

```
setFilter(feature_name,"off")
```

Disables the filter for the specified feature.

```
setFilter(feature_name,"gauss","low","wavelength",3.0)
```

Sets a filter for the specified feature with Gauss, Low-pass, Wavelength Lc 3.0 mm, and Connect Segments = off.

```
setFilter(feature_name,"2 RC","band","undulation",50,300)
```

Sets a filter for the specified feature with 2 RC, Band-pass, Undulations Per Revolution from 50 W/U to 300 W/U, and Connect Segments = off.

```
setFilter(characteristic_name,"featureElement","fromFeature")
```

Sets a filter for the specified characteristic, feature, applying the filter from the feature.

```
setFilter(characteristic_name,"primaryDatum","2
RC","band","undulation",50,300,true)
```

Sets a filter for the specified characteristic with Primary Datum, 2 RC, Band-pass, Undulations Per Revolution from 50 W/U to 300 W/U, and Connect Segments = on.

setGeometryInnerOuter

Defines the feature as an inside or outside feature. The syntax is:

```
setGeometryInnerOuter("Feature ID")
```

This means:

Feature ID	Effect
OUTER	Set to outside feature
INNER	Set to inside feature

This command only acts on the feature in whose input parameters it is contained.

NOTE

A text parameter enables variable specification of feature IDs.

setInspectionStrategy

Defines the current strategy for the entire measurement plan. The syntax is:

```
setInspectionStrategy(StrategyName)
```

This command must be contained in the input parameters of the measurement plan. A blank strategy name will cause selection of the evasion strategy.

setMissingBore

Defines a search distance and reduces the travel speed until reaching the search distance before the first probing. The syntax is:

```
setMissingBore ([feature_name,]search_distance)
```

The feature name can be omitted in the pre-parameters of the current feature.

You will be informed about a missing bore via

```
getActual("feature_name").boreIsMissing
```

The return value is `true` or `false`.

setOutlier

Defines the outlier elimination. The syntax is:

```
setOutlier(measurement_plan_element_name,subelement,factor
i.P.,factor o.P,Area data red.,iterations,filter_criteria)
```

“measurement_plan_element_name” is the name of the feature, the characteristic, or the alignment.

If the first parameter is not the name of a feature, then the second parameter should define the desired subelement.

Value	Meaning
(Undefined)	Feature or feature 1
`"featureElement"`	Feature
`"featureElement1"`	Feature 1
`"featureElement2"`	Feature 2
`"primaryDatum"`	Primary Datum
`"secondaryDatum"`	Secondary Datum
`"tertiaryDatum"`	Tertiary Datum

Value	Meaning
`"rotationSpace"`	Spatial Rotation
`"rotationPlane"`	Planar Rotation
`"translationX"`	X Origin
`"translationY"`	Y Origin
`"translationZ"`	Z Origin

“factor i.P” defines the factor inside the workpiece.

Value	Meaning
Number	Factor for outliers inside the workpiece
`"off"`	Outlier elimination off
`"fromFeature"`	Adoption from feature (only for characteristics)

“factor o.P” defines the factor outside the workpiece.

Value	Meaning
Number	Factor for outliers outside the workpiece
`"off"`	Outlier elimination off
`"fromFeature"`	Adoption from feature (only for characteristics)

“Area data red.” defines the range of data reduction.

Value	Meaning
`"onlyOutliers"`	Only Outliers
`"adjacentPoints"`	Adjacent points i. Number ii. To Computed Feature
`"fromFeature"`	Adoption from feature (only for characteristics)

“iterations” defines the number of iterations.

“filter criteria” followed be two numerical values defines the filter criterion.

Value	Meaning
`"undulation"`	Undulations Per Revolution
`"wavelength"`	Wavelength Lc

Return values

Return value	Meaning
0	Assignment successful
1	Invalid numbers of parameters
2	Elements not found
3	Outlier elimination (for the element) not permitted
4	Invalid data type
5	Invalid value

Examples

```
setOutlier(feature_name,"off")
```

Disables the outlier elimination for the given feature.

```
setOutlier(feature_name,3,3,"onlyOutliers",1,"wave-
length",0,1000)
```

Sets the outlier elimination for the given feature with factors for outliers 3 / 3, Only Outliers, Number of iterations 1, Wavelength Lc from 0 mm to 1000 mm.

```
setOutlier(feature_name,5,5,"adjacentPoints",0,false,1,"undu-
lation",0,4000)
```

Sets the outlier elimination for the given feature with factors for outliers 5 / 5, Adjacent points, Number 0, To Computed Feature OFF, Number of iterations 1, Undulations Per Revolution from 0 W/U to 4000 W/U.

```
setOutlier(characteristic_name,"featureElement","fromFeature")
```

Sets the outlier elimination for the given characteristic, feature, applying the filter from the feature.

```
setOutlier(characteristic_name,"primaryDatum",5,5,"adjacent-
Points",0,true,2,"undulation",0,4000)
```

Sets the outlier elimination for the given characteristic, Primary Datum, with factors for outliers 5 / 5, Adjacent points, Number 0, To Computed Feature ON, Number of iterations 1, Undulations Per Revolution from 0 W/U to 4000 W/U.

setParameterNamed

Assigns a value to a variable during runtime. The syntax is:

```
setParameterNamed(parameter_name,value[,loop index})
```

The parameters have the following meaning:

Parameters	Meaning
Parameter name	Variable with the name of the variable or the variable's name itself
Value	Variable with the value to be assigned or the value itself
Loop index	Variable for the index in the list or the index itself

Example: If the value "Delivery address" with the string `"Main plant Untersiemau"` has been assigned to the Address type variable during runtime, then this results from the command `setParameterNamed(Adress type,Adresse)` assigning the string `"Main plant Untersiemau"` to the `Delivery address` variable.

setRecheckMode

Sets the mode for repeat measurement for the measurement plan. The syntax is:

```
setRecheckMode(Recheck[,Reset][,Maximum])
```

The parameters have the following meaning:

Parameters	Meaning
Recheck = 0	Repeat measurement deactivated
Recheck = 1	Repeat measurement activated
Reset = 0	No reset of part numbers
Reset = 1	All part numbers are reset
Maximum	The maximum number of repeat measurements. 0 = no limitation

setRunID

Sets the mini-plans selected for the CNC start. The syntax is:

```
°setRunID(characteristic group name1,characteristic group
name2, …)
```

The previously saved selection is overwritten. When defining several mini-plans in a call, the total selection forms the scope of measurement.

setShaded

Renders the displayed CAD model. The syntax is:

```
  setShaded()
```

setWireframe

Shows the displayed CAD model as a wire frame model. The syntax is:

```
setWireframe()
```

PCM functions: CMM-specific functions and travel commands

cmmCanUseNavigator

Returns as logical value whether the VAST navigator option can be used. The syntax is:

```
cmmCanUseNavigator()
```

The return value is true if the CMM is suitable for the VAST Navigator option and the VAST Navigator option is available or CALYPSO runs in the Simulation mode. Otherwise, the return value is false.

cncBreak

Cancels the CNC run. The syntax is:

```
cncBreak()
```

The CNC run is broken off, the stop light changes to red.

displayPositionCMM

Opens the **Position** window, in which the current coordinates of the probe are displayed in the machine coordinate system in the same way as with a dial gage.

The syntax is:

```
displayPositionCMM()
```

This dialog allows the user to set each coordinate or all coordinates to “0”. In terms of calculation, this means that the local vector of the position at which zeroing took place is subtracted from the respective current position in the displays in the further course of the process.

After closing with **OK**, the dialog returns the most recently displayed coordinate as the result.

Example: `start_point=displayPositionCMM()`assigns the "start_point“ variable the value `0.005443d@20.345677d@200.311123d`.

getActualSide

Supplies the current clearance plane. The syntax is:

```
°getActualSide()
```

getCNCMode

Supplies the travel mode of the CMM: "manual" or "cnc". The syntax is:

```
getCNCMode()
```

getNextStylusSystemName

Supplies the next required stylus system for stylus system change. The syntax is:

```
getNextStylusSystemName()
```

getPositionCMM

Supplies the current position of the probe in machine coordinates as the result. The syntax is:

```
getPositionCMM()
```

Example: `starting_point=getPositionCMM()` assigns the “starting_point” variable the value `0.005443d@20.345677d@200.311123d`.

getProbe

Returns stylus properties. The syntax is:

```
getProbe("StylusName","PlateName").characteristic
```

Stylus name and plate name must be strings enclosed in straight quotes ("). The possible values of “characteristic” are: „radius“ or „probeName“. The current stylus name can be omitted.

Examples:

Command/assignment	Return value/effect
getProbe().radius	The current probe radius is returned.
getProbe("probe_+Y").radius	The stylus radius of “probe_+Y” is returned.
getProbe("probe_+Y", "plate_A").radius	The radius of “probe_+Y” is returned from “plate_A”.
StylusDiameter=getProbe("probe_+Y", "plate_A").radius	The “StylusDiameter” variable receives the radius of “probe_+Y” from “plate_A”.

getProbeRadiusFromActual

Returns the current stylus radius associated with the indicated measurement in the form of a numerical value in mm. The syntax is:

```
getProbeRadiusFromActual(feature_name[,point_index[,feature_index]])
```

The default value for `point_index` is 1. The Element index is important for patterns.

Examples:

Command/assignment	Return value/effect
getProbeRadiusFromActual("cone1")	Stylus radius of the 1st measuring point of the feature
getProbeRadiusFromActual("cone1",5)	Stylus radius of the 5th measuring point of the feature
getProbeRadiusFromActual("CiRiTop",1,4)	Stylus radius of the 1st measuring point of the 4th feature of the CiRiTop pattern

getRackAssignment

Supplies the active rack assignment. The syntax is:

```
°getRackAssignment()
```

getRTData

Return data to the rotary table. The syntax is:

```
getRTData("parameter")
```

Parameters	Return value
`rtId`	ID of the rotary table When ID missing: empty string, in simulation "0"
`rtAvailable`	true, if rotary table available
`rtActivated`	true, if rotary table enabled
`rtActivatedOrPassive`	true, if rotary table enabled or passive
`passiveRT`	true, if rotary table passive
`mobileRT`	true, if rotary table mobile
`lowerableRT`	true, if rotary table lowerable
`mtmValue`	Current value of the mass moment of inertia in kgm^2
`axisDate`	Date and time of last axis qualification as string
`location`	Position of rotary table in machine coordinate system (vector)
`orientation`	Orientation of rotary table (normal vector in machine coordinate system, vector)
`wobble`	Wobble angle of the RT axis
`eccentricity`	Eccentricity of the RT axis in mm
`angleToMainAxis`	Angle of the RT axis with the main axis

getRTOffset

Returns the rotary table offset. The rotary table offset represents the difference between the relative rotary table position in the measurement plan and the rotary table position in absolute machine coordinates. The syntax is:

```
getRTOffset()
```

getRTPosition

Returns the actual value of the last rotary table position. The syntax is:

```
getRTPosition()
```

(Relative value for features in the measurement plan which can be set)

getRTPositionCMM

Returns the position of the rotary table in absolute machine coordinates. The value cannot be set. The syntax is:

```
getRTPositionCMM()
```

getRunMode

Returns the information about the run state of the CMM The syntax is:

```
getRunMode("parameter")
```

Parameters	Return value true if in the following state:
`inspectionRunning`	Automatic run of a measurement plan
`cmmCNC`	CMM CNC
`autoRun`	AutoRun
`aai`	AAI
`F9Start`	F9 start via control console
`subsequentEvaluation`	Subsequent Evaluation
`simulation`	Simulation
`manCNC`	Manual CNC mode
`planSimulation`	Measurement plan simulation

getTemperatureCorrection

Returns characteristics of the temperature correction object. The syntax is:

```
getTemperatureCorrection().characteristic
```

See the table for the possible values of “characteristic”:

Function	Return value
temperatureCorrection	Boolean value: temperature correction is active or not active
coefficientPart	Thermal expansion coefficient of the component
temperaturePart1	Temperature of workpiece sensor 1
temperaturePart2	Temperature of workpiece sensor 2
temperatureProbe	Temperature of the temperature stylus
temperatureTableFrontBottom	Sensor temperature at the bottom front of the table
temperatureTableFrontTop	Sensor temperature at the top front of the table
temperatureTableRearBottom	Sensor temperature at the bottom rear of the table
temperatureTableRearTop	Sensor temperature at the top rear of the table
temperatureX1	Temperature of sensor 1 on the X scale
temperatureX2	Temperature of sensor 2 on the X scale
temperatureX	Temperature of the scale in the X axis
temperatureY	Temperature of the scale in the Y axis
temperatureZ	Temperature of the scale in Z axis
tempWp	Temperature value of the workpiece used in the CNC run

isManProbeChange

Query for manual stylus system change. The syntax is:

```
°isManProbeChange()
```

Supplies TRUE if manual stylus system change is performed and FALSE in the opposite case.

lowerableRT

Lowers or lifts the lowerable rotary table. The syntax is:

```
°lowerableRT("function")
```

Function	Effect
UP	The LRT is lifted.
DOWN	The LRT is lowered.
STATUS	The status of the LRT is queried. The return value is "UP" or "DOWN".

If the command is used without lowerable rotary table, the error message “9999” will be returned.

measureMTMValue

Performs a measurement of the mass moment of inertia of the loaded rotary table if a suitable rotary table has been installed and logged in. The syntax is:

```
measureMTMValue()
```

This allows you to determine yourself the moment of the mass moment of inertia measurement.

measVolumeIlluminationOff

Deactivates measurement area illumination for O-Inspect. The syntax is:

```
measVolumeIlluminationOff()
```

It makes sense to use this command in the parameters of the feature.

measVolumeIlluminationOn

Activates measurement area illumination for O-Inspect. The syntax is:

```
measVolumeIlluminationOn()
```

It makes sense to use this command in the parameters of the feature.

positionCMM

Moves the probe to the specified position (in machine coordinates). If no axis sequence is specified, the CMM moves in Z first, then Y, and finally X.

This is a direct movement command addressing the CMM. Risk of collision. The probe moves directly to the position; the clearance planes are ignored.

The syntax is:

```
positionCMM(x,y,z,["axis1","axis2","axis3"])
```

The parameters 4 to 6 define the sequence in which the coordinates are executed.

Application: For example, you can define movements to pick-up positions in the output parameters of the measurement plan.

Example: `positionCMM(500,-100,-200)`

The CMM moves first in Z to -200 mm, then in Y to -100 mm and then in X to +500 mm in the machine coordinate system.

Example: `positionCMM(200,-200,-10,"Y","Z","X")`

The CMM moves first in Y to -200 mm, then in Z to -10 mm and then in X to 200 mm.

positionRS

Moves the probe with the specified stylus to the specified coordinates in the applicable alignment. (Standard: Base alignment). The syntax is:

```
positionRS(X,Y,Z,[coordinate_system,stylus])
```

readActualWPTemperature

Reads the current temperature of the specified workpiece sensor in the CNC run and returns it in degree Celsius. The syntax is:

```
readActualWPTemperature([temperature_sensor])
```

Here, “temperature sensor” is a character string comprising one or more sensor numbers connected by "+". Values for “temperature sensor” can be, for example, "1", "2" and "1+2".

Examples:

Command/assignment	Return value/effect
readActualWPTemperature()	The temperature sensors selected in the dialog are read
readActualWPTemperature("1")	Temperature of workpiece sensor 1
readActualWPTemperature("1+2")	Average value of the temperatures of sensor 1 and sensor 2

You can include the read value in the printout or, for example, compare it with the initial value which is available via the `getTemperatureCorrection().tempWp` function.

readWPTemperature

Reads the current temperature of the workpiece sensor and blocks transfer of the temperature immediately before probing the first feature. This requires previous activation of the temperature compensation function. The syntax is:

```
readWPTemperature()
```

If this function is entered in the input parameters of the measurement plan, the temperature transmitted by the sensor will be transferred. This allows the temperature transfer to be brought forward. The function may be used only once in the measurement plan.

rtAxisDefinition

Overwrites the current settings for loading or qualification of the RT axis in the **Rotary Table** window. The syntax is:

```
rtAxisDefinition(rtmode)
```

Values for rtmode: "load" or "measure".

rtPositionOffset

Moves the rotary table to the specified position. The syntax is:

```
rtPositionOffset(offset angle in rad)
```

searchDistance

Defines the probing search path. The syntax is:

```
searchDistance(distance)
```

In this case, “distance” is the distance traveled by the probe before nominal probing. This value must be entered in 0.1 mm. It applies until it is overwritten or canceled by a reset (stop light goes from green to red and then back to green).

Examples:

`searchDistance(10.000)` => Search distance before nominal probing is 1 mm

`searchDistance(60.000)` => Search distance before nominal probing is 6 mm

setCNCMode

Sets the travel mode of the CMM. The syntax is:

```
setCNCMode(mode)
```

Values for mode: "manual" or "cnc".

setInspectionDrivingSpeed

Defines the speed for traveling *between the features*. The syntax is:

```
setInspectionDrivingSpeed(speed)
```

The value for `speed` is given in mm/s.

For safety reasons, it will never be possible to reach a higher speed than the one specified at CNC start, even if you set a higher speed using this function.

This does not affect the speed settings for the individual features.

setMTMValue

Sets the mass moment of inertia of the loaded rotary table to a defined value. CALYPSO will then use the given value. The syntax is:

```
setMTMValue(mtm)
```

Here, "mtm" is the value of the mass moment of inertia in kgm^2.

setNavigationPath

Sets the path from or to the transfer point of the stylus system changer. The syntax is:

```
setNavigationPath(direction,path name)
```

The value for “direction” is either "fromProbeChange" or "toProbeChange". “path name” must result in a valid name of a previously defined path.

If “path name” does not result in a valid path name, CALYPSO will automatically determine the path.

setWPTemperature

Replaces the previously used workpiece temperature value (sensor value or value entered after CNC start) by the specified value. The syntax is:

```
setWPTemperature(temperature)
```

“temperature” is here the temperature value in °C.

The command will only be evaluated if it is used in the measurement plan defaults. It will only take effect if the temperature compensation option has been enabled in the defaults.

Example:

```
// DEFAULT DIALOG
//TEMP_VALUE=inquireNumber("WorkpieceTemp.
// or
// LOAD FILE
ALIST=readListFile("wp_temp_value.txt")
// Access to the 1st value of the list parameter
TEMP_STRING=getParameterNamed(ALIST,1)
// Conversion of string to number
TEMP_VALUE=val(TEMP_STRING)
// DEFAULT OF TEMP. VALUE
setWPTemperature(TEMP_VALUE)
```

stepRS

Moves the probe along X, Y, Z, relative to the current position, in the specified alignment (standard: basic alignment). The syntax is:

```
stepRS(X,Y,Z,[coordinate_system])
```

Commands for intervention in the control

Certain configurations require interventions in the CMM control during the measurement plan run.

For this purpose, PCM offers a special command. If you require this call, you should contact Carl Zeiss, Industrial Metrology Division, for detailed information.

PCM functions: System Commands

actCalypsoWindowName

Returns the name of the CALYPSO window in the Windows operating system. Thus, it can e.g. be minimized by external applications such as HOLOS. The syntax is:

```
actCalypsoWindowName
```

closeSocket

Closes the communication channel between CALYPSO and the external application again. The syntax is:

```
closeSocket()
```

It is used together with the “systemCallForResultAccess” command. It serves to close the “CalypsoInterface” which is used for taking over measurement data from CALYPSO.

For information on using the PCM commands and programming the application that accesses measured data from CALYPSO, see in the operating instructions for the base module under Programmed access to measuring results.

date

Returns the current date in the respective country language. The syntax is:

```
date()
```

Example: `date() = "7 December 2003"`

dateAndTime

Returns system date and time. The syntax is:

```
dateAndTime()
```

Example: date=dateAndTime(). Date and time are written into the date variable (of the “string” type) in the format defined by the system.

dateInNumbers

Returns the current date as numerical values. The syntax is:

```
dateInNumbers()
```

Example: `dateInNumbers() = "12/07/03"`

expandCalypso

Maximizes again the CALYPSO program window or has it displayed in the foreground on the screen. The syntax is:

```
expandCalypso
```

facsIsActive

Returns whether FACS is active. The syntax is:

```
facsIsActive()
```

language

Returns the abbreviation for the currently defined language of the CALYPSO user interface. The syntax is:

```
°language()
```

Example:

For a German user interface, `language()` returns the character string "de".

millisecondClockValue

Accesses the millisecond counter of CALYPSO and returns the current value. The syntax is:

```
millisecondClockValue()
```

This function allows you to measure time in millisecond resolution.

openSocket

Opens a communication channel between CALYPSO and an external application. The syntax is:

```
openSocket()
```

It is used together with the “systemCallForResultAccess” command. It serves to open the “CalypsoInterface” used for taking over measurement data from CALYPSO.

For information on using the PCM commands and programming the application that accesses measured data from CALYPSO, see in the operating instructions for the base module under Programmed access to measuring results.

systemCall

Calls system commands. The syntax is:

```
systemCall("filename")
```

The “filename” file is called and immediately executed. “filename” must be an executable file, e.g. a batch file or a program. You can use this function to copy results to another computer, for example.

Example: `systemCall("D:\philips\main\help.bat")`

The statements in the help.bat file are executed.

NOTE
CALYPSO is not stopped during execution of the system command. Any CNC run will continue.

systemCallForResultAccess

Calls system commands and waits for them to be processed. The syntax is:

```
systemCallForResultAccess("executable command that starts an *.exe")
```

Starts an application. This application can access CALYPSO data while the measurement plan is *not* being processed further. By contrast, “systemCallWithWait” does not permit access to CALYPSO data and “systemCall” does not stop the CNC run.

This command must be used together with the “openSocket” and “closeSocket” commands.

In the example below, commands are sent to CALYPSO by the “SimpleBasicTest” Visual Basic Script via the Weprom interface:

```
openSocket()
systemCallForResultAccess("WScript c:\temp\SimpleBa-
sicTest.vbs")
closeSocket()
```

For information on using the PCM commands and programming the application that accesses measured data from CALYPSO, see in the operating instructions for the base module under Programmed access to measuring results.

systemCallIcon

Calls system commands and waits for them to be processed. The syntax is:

```
systemCallIcon("filename")
```

The “filename” file is called and executed. The executing program is reduced to an icon on the screen. CALYPSO waits until the “filename” file has been processed.

“filename” must be an executable file, e.g. a batch file or a program.

Example: `systemCallIcon("D:\philips\main\help.bat")`

The statements in the help.bat file are executed. CALYPSO will wait until the batch file has been processed.

NOTE
It is not possible to access the measured data of the current measurement plan with this command.

systemCallWithWait

Calls system commands and waits for them to be processed. The syntax is:

```
systemCallWithWait("filename")
```

“filename” file is called and executed in a program window. CALYPSO will wait until the file “filename” has been processed.

“filename” must be an executable file, e.g. a batch file or a program.

Example: `systemCallWithWait("D:\philips\main\help.bat")`

The statements in the help.bat file are executed. CALYPSO will wait until the batch file has been processed.

NOTE
It is not possible to access the measured data of the current measurement plan with this command.

time

Supplies the current time. The syntax is:

```
time()
```

Example: `time() = "9:46:28"`

timeInSeconds

Supplies the current time in seconds. The syntax is:

```
timeInSeconds()
```

Example: `timeInSeconds() = "35246"`

wait

Interrupts the measurement plan run for nSeconds seconds. The syntax is:

```
wait(nSeconds)
```

Example:

```
wait(10)
```

The measurement plan run is stopped for 10 seconds.

PCM functions: Custom printout

getRecHdForFil

Returns the current value of a printout header variable in a form suitable for file names. All characters of the printout header parameter that are not permitted or undesired in file names are replaced with permitted characters. The syntax is:

```
getRecHdForFil("PrintoutHeaderVariable")
```

Example:

The printout header variable “dateshort” has in English the value “3/15/07”. If this character string is used as part of a file name, you will unintentionally create subdirectories.

By accessing the variable via getRecHdForFil, the forbidden characters will be replaced with underscores, thus creating in this case “3_15_07”.

getRecordHead

Returns the current value of a printout header variable. The syntax is:

```
getRecordHead("PrintoutHeaderVariable")
```

If the value of the printout header variable is intended for further use as a file name, the getRecHdForFil command must be used.

getRecordHeadM

Returns the current value of a printout header variable of the corresponding measurement. The syntax is:

```
getRecordHeadM("PrintoutHeaderVariable")
```

presentationOff

Switches off a custom printout. The syntax is:

```
presentationOff()
```

Example: You want a graphic to appear only when certain characteristics are measured. To accomplish this, you define a custom printout with a graphic. You switch off the custom printout by default in the start parameters of the measurement plan. You then request the custom printout for the characteristics in question by opening the **Settings** window and specifying "presentationOn()" in the input parameters.

presentationOn

Switches on a custom printout. The syntax is:

```
presentationOn()
```

setElementsForProtocol

Defines the characteristics to be printed using the following command “displayGraphicsWhileSelection”. The syntax is:

```
setElementsForProtocol
```

setProtocolOutput

Controls the output in printout (listing and printing) independently of the CNC start settings. The syntax is:

```
setProtocolOutput("printout type","output function")
```

- Values for printout type: "compactProtocol", "presentationProtocol", "workingProtocol".
- Values for output function: "list" or "print"

With this command, the CNC start settings will be overwritten and redefined.

setProtocolSetting

Defines the settings for the custom printout. The syntax is:

```
setProtocolSetting("setting","value")
```

The following can be set for “setting” and “value”:

Setting	Value
outputFormat	Name of the desired output format
userDefinedPages	Name of the custom defined page to be attached to the custom printout. Prerequisite: **User Defined Pages** in the **Printout-Output-Definition** dialog box is activated.

Examples:

```
setProtocolSetting("outputFormat","default")
```

defines the “default” format as the output format.

```
setProtocolSetting("userDefinedPages","userProtocol1.gra")
```

attaches the “userProtocol1.gra” page to the existing printout.

setRecordHead

Sets the value of a printout header variable. The syntax is:

```
setRecordHead("PrintoutHeaderVariable","Value")
```

For a complete list of all printout header variables, please refer to the operating instructions for the base module under Reference: Printout header data.

setViewsForProtocol

Defines the CAD views to be output in the printout. The syntax is:

```
setViewsForProtocol(viewname1,viewname2,...,viewnamen)
```

PCM functions: Conditions/Loops

for_next

Defines a certain loop. The defined loop has the following syntax:

```
for index=start to end [step]
   DEFINITION
next index
```

The following must be inserted:

- for `index` - the name of the loop variables (random),
- for `start`, `end` and `step` - whole numbers,
- for `DEFINITION` - random formulas, value assignments, functions or other conditions or loops of your choice, always with line breaks as separators.

Meaning: The functions or commands under DEFINITION are processed (end - start + 1)/step times, whereby “step” is set to 1 if no other specification is made. At the same time, the specific value for index is always entered in DEFINITION: at first start, then start+step, start+2*step etc. up to end. With next i, the loop index is incremented by step each time.

Example:

```
for i=1 to 4
   message(i,". step: ",step[i])
next i
```

if_else_endif

Defines a condition with alternative. The syntax of the condition with alternative is as follows:

```
if CONDITION then
   DEFINITION1
else
   DEFINITION2
endif
```

Meaning:

- If CONDITION is satisfied, DEFINITION1 is processed.
- If CONDITION is not satisfied, DEFINITION2 is processed.

DEFINITION1 and DEFINITION2 can be formulas, value assignments, functions, conditions or loops of your choice, always with line breaks as separators.

Example:

```
message("Test if with PCM")
P1 = 1
message("value is:",P1)
//-----------------------------------------------------------
if P1 == 1 then
   message("is equal. Value was:" ,P1)
endif
if P1 < 1 then
   message("is smaller than 1: Value was:" ,P1)
else
   if P1 > 1 then
      if P1 > 5 then
         message("is greater than 5: Value was:" ,P1)
      endif
   message("is greater than 1: Value was:" ,P1)
   endif
endif
//-----------------------------------------------------------
test = point(1,2,3,0,0,1)
message("X" ,test.x, "Y",test.y, "Z" ,test.z, "nx" ,test.nx,
"ny" ,test.ny, "nz" ,test.nz)
```

if_endif

Defines a simple condition. The syntax of the simple condition is as follows:

```
if CONDITION then
   DEFINITION
endif
```

Meaning:

- If CONDITION is satisfied, DEFINITION is processed.
- If CONDITION is not satisfied, DEFINITION is not processed.

A DEFINITION can be formulas, value assignments, functions, conditions or loops of your choice, always with line breaks as separators.

repeat_until

Defines a conditional loop. The conditional loop has the following syntax:

```
repeat DEFINITION until CONDITION
```

Meaning:

- 1.) DEFINITION is processed. Then CONDITION is tested.
- 2.) If CONDITION is fulfilled, the loop is ended.
- 3.) If CONDITION is not satisfied, DEFINITION is processed again and CONDITION is tested again. Continues with 2.).

The CONDITION can be set up as a logical combination of several subconditions.

The DEFINITION can be formulas, value assignments, functions or other conditions or loops of your choice, always with line breaks as separators.

As the condition is only tested after the first run, a conditional loop must run at least once.

NOTE
Please note that infinite repetitions are possible with the conditional loop if the condition is never fulfilled.

selectCase_endSelect

Defines a condition with several alternatives. The syntax of the condition with several alternatives is as follows:

```
selectCase TEST PRINTOUT
   case VALUE LIST 1
     STATEMENTS 1
   [case VALUE LIST 2
      [STATEMENTS 2]]
   ...
   [caseElse
     [ELSE STATEMENTS]]
endSelect
```

Meaning:

- TEST PRINTOUT:

  Required. Any numeric expression, character string or PCM parameter.

- VALUE LIST:

  Must be specified for `case`.

A list separated by commas that contains one or more of the following forms:

- `Expression`:

  e.g. 12, "All"
- `Expression1 to Expression2`:

  e.g. 1 `to` 15, "Part1" `to` "Part5"
- `is comparative operator expression`:

  e.g. `is` > 8

The operator `to` defines a range of values. The smaller value must be on the left. For a character string, sorting takes place in alphabetical order.

- STATEMENTS:

  Optional. One or more statements that are carried out if TEST PRINTOUT corresponds to any part in the corresponding value list. At the end of `endSelect`, the operation continues. If TEST PRINTOUT corresponds to a value list in several `case` sections, then only the statement after the first match will be executed.
- ELSE STATEMENTS:

  Optional. One or more statements that are carried out if no match between TEST PRINTOUT and a value list is found in the `case` sections.

NOTE

Conditions with several alternatives can be nested.

For example, a condition with several alternatives can look as follows:

```
selectCase number
      case is <= 0
            Message = "Number is too small"
      case 1, 2, 3
            Message = "Number is smaller than 4"
      case 4 to 7
            Message = "Number is between 4 and 7"
      case is >8 and number < 11
            Message = "Number is 9 or 10"
      caseElse
            Message = "Number is outside the range"
endSelect
```

PCM functions: Output in printout

defineProtocol

Used for the definition of the output. The syntax is:

```
defineProtocol (printout parameter,value)
```

Several values can be entered for the "#medium" output parameter:

```
defineProtocol ("#medium",value1, ... ,valuen)
```

For the complete definition of an output, use several lines with this command and define one output property on each line. Example:

```
defineProtocol("#name","Test PP1")
defineProtocol("#type","presentationProtocol")
defineProtocol("#medium","printer")
defineProtocol("#elements","allCF")
defineProtocol("#order","likeList")
```

NOTE

The “defineProtocol” PCM function disables the set printout via **Resources → Define Printout**.

NOTE

This function is suitable for use in external PCM files. Thus you can simplify the PCM printout for several measurement plans.

Possible function values per output parameter

You can assign the following values to the output parameters:

Parameter	Possible values		
"#name"	User-definable character string		
"#type"	"compactProtocol"	"presentationProtocol"	"workingProtocol"
"#format"	"<format name for printout header>"	"<format name>"	–
"#elements"	"allME" "allMENom" "allCF" "allCFLimit" "allCFOutTol"	"allCF" "allCFLimit" "allCFOutTol"	–

Parameter	Possible values		
"#order"	"alphabetic" "elementType" "rangeOfTolAndDev" "likeRun"	"likeList" "alphabetic" "elementType" "rangeOfTolAndDev" "likeRun"	-
"#medium"	"screen" "printer" "pdf" "ps"	"screen" "printer" "pdf" "ps"	"screen" "printer" "pdf" "ps"°
"#warningLimitCF"	Percentage		

Meaning of the individual function values

The table contains the meaning of the individual function values.

Function value	Meaning
compactProtocol	Compact printout
presentationProtocol	Custom Printout
workingProtocol	Default printout
allME	All features and all characteristics
allMENom	Only features with characteristics
allCF	All Characteristics
allCFLimit	Characteristics from warning limit
allCFOutTol	Characteristics outside tolerance
likeList	Like Program
alphabetic	Alphabetic
elementType	By Type
rangeOfTolAndDev	Positive To Negative
likeRun	After run
screen	Display
printer	Printer
pdf	PDF output
ps	PostScript

Function of the defineProtocol command

The line with the “#name” output parameter defines the output action. As long as no other name has been defined, all following defineProtocol functions have an effect on the so-called output action.

If the definition of an output action does not contain all possible functions, the last defined value applies or if no value has been defined, the value set under **Resources → Define printout** applies.

Thus you can create several outputs on the same output medium with different formats and output scopes. Example:

```
// 1st output action
defineProtocol("#name","Test PP1")
defineProtocol("#type","presentationProtocol")
defineProtocol("#medium","printer")
defineProtocol("#elements","allCF")
defineProtocol("#order","likeList")
// 2nd output action
defineProtocol("#name","Test PP2")
defineProtocol("#order","rangeOfTolAndDev")
defineProtocol("#type","presentationProtocol")
defineProtocol("#medium","printer")
defineProtocol("#elements","allCFOutTol")
```

These PCM functions create two outputs: The custom printout is printed twice.

In the first printout (“Test PP1”), all characteristics ("#elements","allCF") are output in the order of the list ("#order","likeList"). In the second printout (“Test PP2”), the characteristics which are out of tolerance ("#elements","allCFOutTol") are output in the order of the tolerance exceeding ("#order","rangeOfTolAndDev").

Overview: Names of external PCM files

External PCM files are executed automatically in certain measurement plan run situations if they have a certain name and are saved in a certain directory.

Files in ...	are executed ...
directory of a measurement plan *<user_directory>\workarea\inspections\<measurement_plan_name>*	during the run of the corresponding measurement plan
global measurement plan directory *<user_directory>\workarea\inspections*	during the run of each measurement plan

The name of the external files defines where in the measurement plan run the file is executed.

Execution	PCM file name
Prior to the start dialog	inspection_pre_start_dialog_pcm.txt
After loading the measurement plan	inspection_post_load_pcm.txt
At the beginning of the run	inspection_start_pcm.txt
After the end of the measurement; during the run after features prior to the computation of characteristics	measurement_end_pcm.txt
Prior to the execution of the input parameters of each feature	plugin_preFeature_pcm.txt
Prior to the execution of the output parameters of each feature	plugin_postFeature_pcm.txt
Prior to the execution of the input parameters of each characteristic	plugin_preCharacteristic_pcm.txt
Prior to the execution of the output parameters of each characteristic	plugin_postCharacteristic_pcm.txt
After the end of the computation, but prior to the report/file output (not during the run with a selection of features)	calculation_end_pcm.txt
After the end of the run and the report output	inspection_end_pcm.txt
After completion of all (also externally) created reports and files if they are started by CALYPSO (e.g. PDF, DFD/DFX) (not during the run with a selection of features)	report_end_pcm.txt
Prior to saving the measurement plan	inspection_pre_save_pcm.txt

Alphabetic index