sonar

sonar Script Language Guide

Version 2.2
Status : for general use
Author : Fritz Leibundgut, L&G Software
© L&G Software 2017

Inhalt

Sprache 8

Rohdaten 19

Primitivkörper 27

Eigenschaften (Primitives) 38

Objekt-Interaktion 47

Variablen (Primitives) 51

Operationen an Primitives 58

Links 63

Fixpunkte 69

Gruppeneigenschaften 71

Materialmodelle 74

Kontrollsysteme 79

Uebergeordnete Eigenschaften 82

Halbfabrikate 84

Uebersicht ‘Macro Language’ in alphabetischer Reihenfolge 88

Uebersicht Macro Language nach Funktionsgruppen 94

Kontrollsystem Sprache 102

Reservierte Worte 104

Sprache

Einführung

sonar script hat die Aufgabe ein komplettes sonar-Simulationsmodell mittels niedergeschriebenen Anweisungen zu erzeugen. Statt dass der Benutzer ein Modell mit Hilfe von grafisch-interaktiven Werkzeugen und Funktionen direkt am Bildschirm zeichnet, definiert er denselben Vorgang mit einer Reihe von Anweisungen, welche im Prinzip genau die gleichen Handlungen in Textform festhalten. Das gesamte Modell, welches der Benutzer manuell am Bildschirm zeichnen könnte, wird stattdessen in Textform in einer Datei gespeichert. Dieses Vorgehen hat gegenüber der klassischen Definition oft den Vorteil, dass Aenderungen oder Varianten eines Basismodells schneller in einem Makro durchgeführt werden, statt an einem fertigen sonar-Modell. Ein sonar script ist auch eine hervorragende Dokumentation eines Modells, weil in ihm sämtliche geometrischen und physikalischen Einstellungen in schriftlicher und übersichtlicher Form festgehalten werden.

sonar script' ist eine universelle interpretierende Makro-, Kontrollsystem- und Kommandosprache in Zusammenhang mit der sonar-Software. 'interpretierend' heisst, dass die Sprache zum Zeitpunkt der Ausführung von einem Compiler übersetzt bzw. interpretiert wird. 'sonar script' wird folglich nicht in eine Maschinensprache übersetzt bevor sie zur Ausführung gebracht wird. Die sonar script-Sprache beinhaltet einen Scanner, einen Parser, einen Interpreter und eine Syntaxanalyse. Wird ein sonar-script zur Ausführung gebracht, dann wird das gesamte Script zuerst virtuell ausgeführt und auf Fehler kontrolliert. Werden bei dieser Kontrolle keine Fehler entdeckt, dann wird das Script effektiv zur Ausführung gebracht.

sonar Script files sind im Text-Format geschrieben, haben die Endung (.txt) und können mit jedem textverarbeitenden Programm erstellt und verändert werden. Besonders geeignet ist dafür das Windows-Systemprogramm 'Editor' welches auf jedem Windows Computer zur Verfügung steht.

sonar Script ist eine Sprache die evolutionär entstanden und gewachsen ist. Der Sprache lag ursprünglich kein Plan zugrunde, welche reservierten Worte in den Sprachschatz der Sprache aufgenommen werden sollen. Die Sprache entstand und wuchs parallel während der Entwicklung der Software. Die Sprache hatte aber von Anfang an das Konzept, einfach zu bleiben und wenn immer möglich die gleiche Satzstruktur zu verwenden. Es bestand die Absicht eine sehr einfache, leicht zu erlernende Sprache zu definieren, welche jedermann, auch Nicht-Informatiker leicht verstehen kann. Es sollte eine Sprache sein, welche gut lesbar und wenn möglich selbsterklärend ist. In der Tat hat sonar script wenig zu tun mit einer ausgewachsenen Programmiersprache. sonar script hat, wenn überhaupt, eher etwas zu tun mit Kommandosprachen (command language) welche Computer auf einer übergeordneten Ebene steuern. Der Sprachschatz von sonar script wird in absehbarer Zeit auch nicht abgeschlossen sein. Oft, wenn neue Funktionen in sonar implementiert werden, sind auch neue Sprachkonstrukte notwendig um diese Vorgänge in script-Form festzuhalten.

sonar script ist in einem gewissen Sinn universell einsetzbar. In der sonar-Umgebung wird die Sprache für verschiedene Zwecke eingesetzt:

- Makrosprache
- Control System Sprache
- Kommandosprache

Diese verschiedenen Anwendungsgebiete verwenden teilweise eigene sachbezogene Wörter, aber die Grundstruktur der Sprache ist dieselbe.

Nomenklatur

In diesem Buch werden zur Unterscheidung verschiedener Sprachelemente und zum besseren Verständnis unterschiedliche Schrifttypen verwendet. Es gelten die folgenden Formatierungsregeln:

`LANGUAGE ELEMENT`

Reservierte Worte und entsprechende Anweisungen in der 'sonar script'-Sprache werden mit der Schrift `COURIER` dargestellt. Die Gross- und Kleinschreibweise in dieser Schrift kann jedoch beliebig verwendet werden.

placeholder

Ein Platzhalter muss vom Benutzer durch ein reserviertes Wort aus einer Auswahl von mehreren möglichen Worten ersetzt werden. Im weiteren werden die folgenden speziellen Platzhalter verwendet:

bool : `TRUE, FALSE, YES, NO, ON, OFF`
int, integer : ganzzahliger Zahlenwert
float, double : (Fliesskommawert in einfacher oder doppelter Präzision)

`[optional]`

Rechteckige Klammern zeigen an, dass die darin enthaltenen Sprachelemente nicht zwingend, sondern optional sind.

`a | b | c`

Senkrechte Trennstriche zwischen Sprachelementen haben die Bedeutung des Wortes 'oder'. Der Benutzer ist aufgefordert eines der aufgelisteten Worte auszuwählen.

Sprachaufbau

Zu Beginn ein paar sonar script Anweisungen:

```
CREATE OBJECT (O1, SPHERE, 1,  1,  5,  2.4)
SET PROPERTY (O1, DENSITY, 7.8)
CREATE FIELD (GRAVITATION, 0, -1, 0, 9.81E-10)
```

Jeder Leser versteht, zumindest dem Sinn nach, was diese Anweisungen ungefähr bedeuten könnten.

- Es wird ein Object vom Typ Kugel erzeugt
  und zwar an der Position (1, 1, 5)
  und mit dem Radius von 2.4
- Die Dichte dieses Objektes (O1) wird auf 7.8 festgesetzt

- Es wird ein Gravitationsfeld eingeschaltet
  in Richtung (0, -1, 0)
  und mit einem bestimmten Wert für die Gravitationsbeschleunigung.

Die Klammerausdrücke (1, 1, 5) und (0, -1, 0) sind Vektoren (x, y, z).

Aus diesen Beispielen wird bereits der grundsätzliche Aufbau von sonar script Anweisungen deutlich. Dieser Aufbau zieht sich wie ein roter Faden durch die gesamte sonar script Sprache hindurch

Command Qualifier (parameterList)

```
Was tun ? ..... ( mit Was und Wie genau ? ..........)
```

Aus einer Reihe von Anweisungen (statements) dieser Form lassen sich ganze Anweisungsblöcke bzw. Anweisungslisten zusammenstellen.

statementList = statement
statement
statement
. . . . . .
statement

bekommt die Anweisungsliste nun noch eine Kopflinie, dann haben wir ein 'sonar script'.

```
BEGIN SCRIPT scriptName
    statementList
```

Dies ist ganz grob der Aufbau der sonar script Sprache. Eine sehr einfache und pragmatische Sprache ohne Schnörkel, welche minimal und ohne Umwege sagt was geschehen soll.

Anweisung (statement)

sonar script ist eine Linien-orientierte Sprache. Jede Anweisung endet mit einer Zeilenschaltung (return). Umgekehrt markiert jede Zeilenschaltung das Ende einer Anweisung. Die maximale Länge einer Anweisung beträgt 256 Zeichen. Genau genommen sind es 255 Zeichen plus die Zeilenschaltung. Wird eine Programmzeile länger, dann wird der Ueberhang vom Interpreter abgeschnitten und was übrigbleibt erzeugt wahrscheinlich eine Fehlermeldung (syntax error). Die Anzahl Anweisungen die ein script haben darf ist nicht begrenzt.

Kommentare (comments)

Wie in jeder Computersprache gibt es auch in sonar script die Möglichkeit, den Text eines scipts mit Kommentaren zu versehen. Diese Kommentare nehmen an der eigentlichen Ausführung nicht teil, sondern sind Randbemerkungen des Autors für den Leser. Es sind Erklärungen die dem Verständnis des Programms dienen. Nicht zuletzt sind es Erinnerungen an sich selbst, damit man nach einem Jahr noch weiss, was die Anweisungen bewirken sollen. Kommentare sind freiwillig, aber sehr empfohlen. Die Eingangs angeschriebenen beispielhaften Anweisungen lassen sich mit dem bisher gesagten wie folgt zu einem vollständigen script erweitern.

```
BEGIN SCRIPT myScript
-- eine Stahlkugel im freien Fall
-----------------------------------------------
CREATE OBJECT (O1, SPHERE, 1,  1,  5,  2.4)
SET PROPERTY (O1, DENSITY, 7.8) -- Dichte von Stahl
CREATE FIELD (GRAVITATION, 0, -1, 0, 9.81E-10)
-- end of script
```

Zwei Minuszeichen bezeichnen das Ende einer Anweisung. Alles was nach den beiden Minuszeichen bis an das Ende der Linie noch kommt, wird ignoriert. De Facto werden in einer Anweisung die beiden Minuszeichen und der Rest der Linie abgeschnitten bevor die Linie interpretiert wird. Kommentare dürfen folglich am Ende einer Anweisung angefügt werden oder es dürfen auch eigentliche Kommentarlinien wie im Beispiel oben verwendet werden. Beachten Sie, dass ein script auch leere Linien haben darf.

Gross- und Kleinschreibung

sonar script ist nicht sensitiv bezüglich der Gross- und Kleinschreibung. Die Gross- und Kleinschreibung kann beliebig gemischt werden. Wörter die sich nur in der Gross- und Kleinschreibung unterscheiden sind gleichwertig:

```
'DENSITY' = 'density' = 'dEnSiTy'
'O1' = 'o1'
'9.81E-10' = '9.81e-10'
usw.
```

Die Gross- und Kleinschreibung kann als weiteres Mittel zur Strukturierung eines scripts benutzt werden. Eine häufig angewandte, gute Methode ist, alle reservierten Worte des Systems in Grossbuchstaben anzuschreiben und eigene Namensgebungen bzw. eigener Text mit der Kleinschreibung zu verwenden.

Objekt Referenz (object reference)

Wir erzeugten im letzten script eine Kugel mit der Anweisung
`CREATE OBJECT (O1, SPHERE, 1,  1,  5,  2.4)`
Der erste Parameter in dieser Anweisung heisst 'O1'. Diese Abkürzung bedeutet 'Object 1'. Zur Erzeugung einer Kugel ist dieser Parameter an sich nicht notwendig. Dieser Parameter hat viel mehr den Zweck eine Referenz auf das soeben zu erzeugende Objekt zu schaffen. Wenn wir im weiteren Verlauf an einem bestimmten Objekt noch zusätzliche Einstellungen vornehmen oder weitere Eigenschaften setzen wollen, dann müssen wir in der Lage sein, dem System mitzuteilen welches Objekt wir damit meinen. Wir benötigen eine Referenzierung bzw. einen Namen für das betreffende Objekt. Und genau dieser Name setzen wir beim Erzeugen der Kugel in obiger Anweisung. Wir nennen die Kugel 'O1' oder 'O16'. In der zweiten Anweisung des letzten scripts, wo wir die Dichte setzten, benutzten wir alsdann diese Referenz und sagten: "Setze die Dichte der Kugel mit dem Namen 'O1' auf einen bestimmten Wert.
Eine Objektreferenz gehorcht dem Syntax: 'O' & *integer*, also

O1, o2, o16, ...

Die ganze Zahl muss in folgendem Zahlenbereich liegen: [1..16383]. Genauso wie es Referenzierungen für Objekte gibt, gibt es auch welche für andere Typen:

```
Elemente:    E1, e2, ...
Konturen:    C1, c2, ...
Objekte:     O1, o2, ...
Links:       K1, k2, ...
Fixpunkte:   F1, f2, ...
Grid's:      G1, g2, ...
Aktuatoren:  A1, a2, ...
Daempfer:    D1, d2, ...
```

Das Einheitensystem

Die sonar-Software rechnet im sog. [cm-g-µs]-System. In Worten: im Zentimeter-Gramm-Mikrosekunden-System. Weshalb dieses ungewöhnliche Einheitensystem verwendet wird, hat etwas zu tun mit der numerischen Integration der Basisgleichungen in sonar während der Simulation (Sehen Sie dazu das sonar-Tutorial). Aus diesem Grund müssen in einem 'sonar script' alle verwendeten Variablen in diesem Einheitensystem verwendet werden. Es besteht die Absicht, dem Benutzer ein Tool in Form eines Dialogs zur Verfügung zu stellen um bestimmte Zahlenwerte 'ad hoc' vom metrischen in dieses 'calculation' System umzurechnen.

Für's Erste wollen wir uns hier mit einem Beispiel begnügen. Im letzten script-Beispiel benutzten wir die folgende Linie:
`CREATE FIELD (GRAVITATION, 0, -1, 0, 9.81E-10)`
Als letzter Parameter kommt in dieser Linie die Gravitationsbeschleunigung auf der Erdoberfläche mit 9.81 m/s^2 vor. Die Umrechnung in das sog. 'calculation'-System geschieht in diesem Fall wie folgt:

$$9.81\ m/s^2 = 9.81 * 1E+2\ cm / (1E+6\ us)^2 = 9.81E-10\ cm/us^2$$

und dieser Wert wurde in unserem script folglich verwendet.

Ein script öffnen und ausführen

Ein script wird in einem Textverarbeitungsprogramm geschrieben und vorbereitet (z.B. im Systemprogramm EDITOR). Gespeichert wird das script als Textfile mit der Endung (.txt)

Zurück in sonar_LAB befindet sich am rechten Rand des Bildschirms das 'MacroTool' mit einem eigenen Menu. Mit Hilfe dieses Menus öffnen wir unter Verwendung des Standard File Dialogs das vorbereitete script mit

```
MacroTool / File / Open Macro
```

Ebenfalls über das Menu im Macro Tool sind wir in der Lage das script zur Ausführung zu bringen:

```
MacroTool / Macro / Execute
```

Als Erstes überprüft das System allerdings das Makro auf mögliche Syntaxfehler und beginnt mit der Ausführung des scripts erst dann, wenn der Text diese Prüfung besteht. Andernfalls wird eine Fehlermeldung angezeigt und ein Hinweis auf den Ort und die Ursache des Fehlers gemacht. Diese Vorgehensweise schützt uns davor, dass das Programm mit der Erzeugung von Objekten beginnt, hängen bleibt und uns eine unfertige Baustelle des

angestrebten Modells hinterlässt welches wir ohnehin nicht weiterverarbeiten können und wollen. Dies wäre besonders dann hinderlich, wenn mehrere scripts hintereinander aufgerufen und ausgeführt werden und plötzlich stellt ein script die ganze Arbeit in Frage.

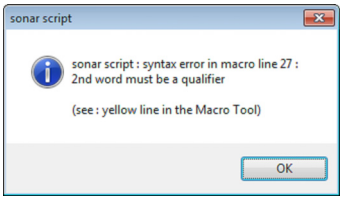


FIGURE 1. Eine Fehlermeldung nach der Auslösung des Menu-Befehls 'Execute' .....

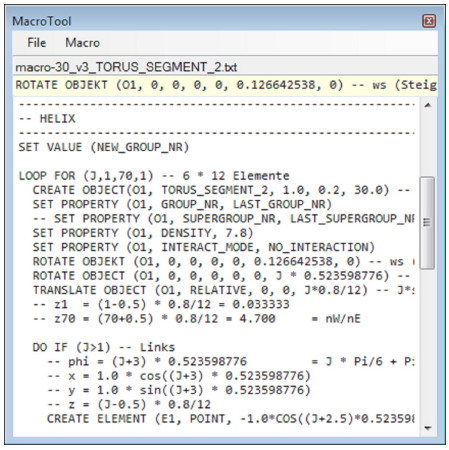


FIGURE 2. ....., und die Programmlinie welche den Fehler verursacht hat, angezeigt im Kopfteil des Macro-Windows (gelbe Linie)

Wie in der sog. gelben Linie angezeigt, wurde das zweite Wort in der betreffenden Programmlinie als 'OBJEKT' statt als 'OBJECT' geschrieben.

Ausdrücke (expressions)

Ausdrücke sind Anweisungen die ein Resultat liefern. Im Gegensatz zu einer Anweisung der Form `'SET PROPERTY (O1, DENSITY, 7.8)'`, wie wir sie bereits benutzt haben, gibt es Ausdrücke der Form

```
POS.X(O14) - POS(O3) + 1.2          (expression)
```

oder

```
POS.X(O14) < 12.5                   (boolean expression)
```

Der erste der beiden Ausdrücke liefert als Resultat einen Zahlenwert, der zweite ein Ergebnis der Form 'wahr' oder 'falsch' (TRUE, FALSE), je nachdem welchen Wert die X-Komponente der Position von Objekt Nr.14 im Moment gerade hat. Dieser Wert kann sich im Laufe einer Simulation natürlich ändern. Deshalb kann man mit einem sog. Bool'schen Ausdruck das Eintreten eines bestimmten Ereignisses abfangen.

Bedingte Anweisungen (conditional statements)

Innerhalb eines scripts kann die Ausführung einzelner Linien oder ganzer script-Segmente von gewissen Bedingungen abhängig gemacht werden. Die Bedingungen sind sog. Bool'sche Ausdrücke welche ein Resultat der Form (TRUE oder FALSE) liefern. Nur wenn dieser Test wahr ist werden die nachfolgenden umschlossenen Anweisungen ausgeführt. Die Syntax einer bedingten Anweisung sieht wie folgt aus:

Syntax:

```
DO IF (boolean expression)
   statementList
END IF
```

'DO IF' und 'END IF' sind reservierte Wortkombinationen. Bedingte Anweisungen können beliebig verschachtelt werden, d.h. die bedingten Anweisungen können ihrerseits wieder Bedingungen enthalten.

Beispiel:

```
DO IF (T < 0.5E+6) -- falls die Zeit T < 0.5s
   statementList
   DO IF (POS.X(o3) <= 10.2)
      statementList
   END IF
   statementList
END IF
```

Jede 'DO IF' Anweisung muss mit einer 'END IF' Linie abgeschlossen werden, d.h. die Anzahl der 'DO IF'- und 'END IF'-Anweisungen in einem script ist gleich. Andernfalls wird eine Fehlermeldung angezeigt.

Schleifen (Loops)

'Loops' sind ein wichtiges und unentbehrliches Mittel in einer Programmiersprache. Mit 'Loops' sind wir in der Lage, sich wiederholende script-Abschnitte mehrmals durchlaufen zu lassen und z.B. eine Kette von sich wiederholenden Teilen zu erzeugen. Mit verschachtelten Loops generieren wir einen dreidimensionalen Kugelhaufen, usw.

Syntax

```
LOOP FOR loopControl
    statementList
END FOR
```

Die sog. '*loopControl*' bestimmt darüber, wie oft und mit welchen Indizes der Loop durchlaufen wird. Kommt die Ausführung gemäss diesen Vorgaben schliesslich an ein Ende, dann wird der nächste script-Abschnit, welcher der Loop-Struktur nachfolgt, ausgeführt. Die '*loopControl'* hat die folgende Syntax:

Syntax

loopControl := (*loopVariable*, index start, index end, index step)

Das erste Mal wenn der Loop beginnt, wird die *loopVariable* auf den Wert *start* gesetzt. Anschliessend wird die *loopVariable* mit jedem Durchgang um den Wert *step* erhöht. Wird der Wert *end* überschritten, wird der Loop abgebrochen. Am besten zeigt man das Ganze an einem Beispiel:

Beispiel

```
LOOP FOR (J, 0, 12, 1)
    statementList
    LOOP FOR (I, 1, 8, 2)
        statementList
    END FOR
    LOOP FOR (I, 0, 20, 1)
        statementList
    END FOR
    statementList
END FOR
```

Zu diesem Beispiel gibt es ein paar Bemerkungen zu machen.

- Der sog. J- Loop mit der *loopVariable* 'J' wird insgesamt 13 mal durchlaufen
- Der erste I-Loop hingegen wird nur 4 mal abgearbeitet, und zwar der Reihe nach für die Indizes (1, 3, 5, 7)
- Der zweite I-Loop, welcher 21 mal durchlaufen wird, verwendet die gleiche *loopVariable* 'I' wie der vorangehende Loop. Das ist zulässig, solange sich die Loops nicht gegenseitig in die Quere kommen und redundant werden. So dürfte man anstelle des zweiten I-Loopes z.B. die *loopVariable* 'J' nicht nochmals verwenden, denn die J-Variable ist an dieser Stelle immer noch aktiv und noch nicht fertig abgearbeitet.
- Die *loopVariablen* dürfen innerhalb der *statementList* wieder als Variable verwendet werden. So könnte es z.B. innerhalb eines 'I'-Loops eine Anweisung geben der Form:

  ```
  CREATE OBJECT (O1, SPHERE, I*2, J*2, 0, 0.3)
  ```

- Die Multiplikation der X- und Y-Koordinaten der Kugeln mit den *loopVariablen* bewirkt, dass ein reguläres 2-dimensionales Netz von Kugeln entsteht.

Funktionen

In der sonar script Sprache sind die folgenden Funktionen implementiert

mathematische Funktionen

```
ABS (expression)    Absolutbetrag
SQR (expression)    Quadrat
SQRT (expression)   Quadratwurzel
EXP (expression)    Exponentialfunktion
LOG (expression)    der natürliche Logarithmus
LOG10 (expression)  der Logarithmus zur Basis 10
```

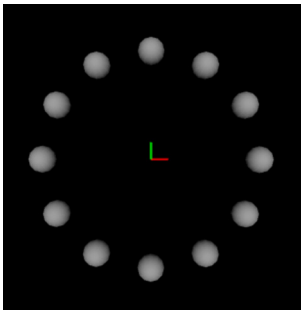
Trigonometrische Funktionen

```
SIN (expression)    Sinusfunktion
COS (expression)    Cosinusfunktion
TAN (expression)    Tangensfunktion
ASIN (expression)   Arcussinus Funktion
ACOS (expression)   Arcuscosinus Funktion
ATAN (expression)   Arcustangens Funktion
```

Bool'sche Funktionen

```
AND                 logische UND-Verknüpfung
OR                  logische ODER-Verknüpfung
NOT                 logische NICHT-Verknüpfung
```

Beispiel:



```
LOOP FOR (J, 1, 12, 1)
   -- ein Kreis von 12 Kugeln
   -- Radius = 4
   -- 3.14 = π
   -- die folgende Anweisung ist auf einer Linie geschrieben.
   CREATE OBJECT (O1, SPHERE, 4*COS(J*3.14/6),
                              4*SIN(J*3.14/6), 0, 0.5)
END FOR
```

Bemerkungen

- Die trigonometrischen Funktionen nehmen als Argument einen Winkel im Bogenmass entgegen und nicht in Winkelgrad.
- Zum obigen Beispiel: Da die SIN- und COS-Funktion immer eine Zahl im Wertebereich von [-1 .. +1] zurückgibt, ergibt die Multiplikation dieser Funktion mit der Zahl '4' den Radius des Kreises, bzw. die '4' ist der Radius des Kreises. Das Winkelargument der SIN- und COS-Funktion enthält den Indexzähler 'J'. Daraus folgt, dass der Winkel in der betr. Klammer mit jedem Durchgang des Loop's entsprechend erhöht wird. Und zwar jedesmal um $\pi/6$. Wenn der Loop also 12 mal durchlaufen wird, ergibt der Winkel beim letzten Durchgang = $12 * \pi/6 = 2\pi$. Und das wiederum ist gerade ein voller Kreis. Deshalb werden die 12 Kugeln auf einen ganzen Kreis verteilt.
- An dieser Stelle kommt eine wichtige Frage auf. In einem 'Loop' wie oben im letzten Beispiel angeschrieben wird in jedem Durchlauf eine neue Kugel erzeugt und jedesmal bekommt diese Kugel den Namen bzw. die Objekt-Referenz 'O1'. Soll das heissen, dass diese Kugeln alle den gleichen Namen haben? Die Antwort lautet: Nein, immer die letzte Kugel die unter der Bezeichnung 'O1' erzeugt wurde heisst so. Alle anderen Kugeln verlieren ihren Namen und sind in der weiteren Folge des scripts nicht mehr auf diese Weise referenzierbar. Will man also jeder Kugel in

diesem ‘Loop’ noch gewisse Eigenschaften geben, dann macht man das innerhalb des Loops, direkt nachdem eine Kugel erzeugt wurde und ihre Referenz noch gültig ist. De Facto funktioniert das Ganze in der sonar script Software so, dass intern eine Objektliste mit zugehörigen Objekt-Nummern angelegt wird. Jedes erzeugte Objekt bekommt einen Eintrag wo genau steht, welche Objekt-Nummer zu Objekt ‘O1’ gehört, usw. Wird nun eine neue Kugel mit der Referenz ‘O1’ erzeugt, dann wird einfach der alte Eintrag mit der neuen Objektnummer überschrieben. Wieder auf das letzte Beispiel bezogen steht am Schluss, wenn der Loop an sein Ende gekommen ist, in dieser Objektliste die Objekt-Nr. 12 neben dem Objekt-Namen ‘O1’.

Operatoren

Die Sprache ‘sonar script’ kennt die folgenden Operatoren:

TABLE 1. Operatoren

Operator	Operation, Typ	Resultat
+	Addition	Zahlenwert (double)
–	Subtraktion	Zahlenwert (double)
*	Multiplikation	Zahlenwert (double)
/	Division	Zahlenwert (double)
<	kleiner	Booelan (TRUE, FALSE)
<=	kleiner oder gleich	Booelan (TRUE, FALSE)
>=	grösser oder gleich	Booelan (TRUE, FALSE)
>	grösser	Booelan (TRUE, FALSE)
==	gleich, identisch	Booelan (TRUE, FALSE)
<>	ungleich	Booelan (TRUE, FALSE)
()	Klammerausdrücke	werden zuerst ausgewertet
=	Anweisung	Wert Zuweisung an Variable

Operatoren Rangfolge

Soweit nicht mit Klammerausdrücken näher spezifiziert werden die Operationen in einem Ausdruck nach einer bestimmten Rangfolge ausgeführt. Nach ihrer Priorität geordnet ist das die folgende Rangfolge:

TABLE 2. Rangfolge der Ausführung

Rangfolge, Priorität	Operator	Operation, Typ
1	()	Klammerausdruck
2	–	negatives Vorzeichen
3	* /	Multiplikation, Division
4	+ –	Addition, Subtraktion
5	< <= >= >	Vergleich
6	== <>	Vergleich

Beispiel

```
2*4+8/4 = 10
2*(4+8)/4 = 6
```

Eine spezielle Variable

Wir haben gelernt, wie man mit sog. Objekt-Referenzen andere Objekte, die man vorher im gleichen Script erzeugt hatte, referenzieren und wiederverwenden kann. Darüber hinaus haben wir gesehen, dass Objekte allenfalls auch durch eine Selektion an einem bestimmten Raumpunkt selektiert und weiterverwendet werden können. Hier kommt nun eine weitere Methode hinzu, andere Objekte zu selektieren um sie in Funktionen wiederzuverwenden, auch wenn diese in einem anderen script oder manuell erzeugt wurden. Es ist die Selektion des letzten erzeugten Objektes. Dabei ist es völlig egal, wie dieses erzeugt wurde. Es wird einfach das letzte Objekt im Objekt-Speicher, so wie es auch im sog. 'Object-Tool' im Objekt-Ordner eingetragen ist, selektiert.

Syntax

```
SELECT OBJECT (LAST_OBJECT)
```

Die Funktion lässt auch zu, dass man Objekt-Nummer noch modifiziert, indem man eine gewisse ganze Zahl subtrahiert:

Syntax

```
SELECT OBJECT (LAST_OBJECT - n)
```

Zum Beispiel selektiert die folgende Anweisung das drittletzte Objekt in der aktuellen Objektliste.

```
SELECT OBJECT (LAST_OBJECT - 2)
```

sonar script Limits

Die folgende Tabelle hält ein paar Grenzwerte der sonar-script Implementierung fest.

TABLE 3. sonar script limits

max. statement length	255 characters
max. script lemgth	unlimited (capacity of a Windows textfield)
max. number of references of a type	16383
max. number of nested loops	
max. number of parameters in a script	
max. number of open scripts	1
max. number of open models	1

Rohdaten

Einführung

Rohdaten sind meistens 2-dimensionale, in einzelnen Fällen auch 3-dimensionale Zeichendaten. 2-dimensionale Rohdaten werden in den häufigsten Fällen in der X-Y-Ebene (Front View) erzeugt. Die Z-Koordinaten dieser Daten werden dann durchwegs auf Null gesetzt. Dies geschieht in diesem Sinne auch in den folgenden Beispielen.
Auch importierte Rohdaten aus CAD-Programmen (z.B. DXF Daten) sind reine Zeichendaten welche noch keine physikalische Bedeutung haben. Erst durch die Weiterverarbeitung der Rohdaten zu 3-dimensionalen physikalischen Objekten werden diese schliesslich zu dem was wir in einer Simulation benötigen. In sonar kennen wir die folgenden Rohdaten-Elemente:

- POINT
- LINE
- ARC
- CIRCLE
- POLYGON
- POLYLINE
- QUADSTRIP

Diese Basiselemente können so weit möglich zu komplexeren Strukturen zusammengesetzt werden. Insbesondere machen wir in 'sonar' oft Gebrauch von sog. LINE-ARC-Konturen.

Point

```
CREATE ELEMENT (E1, POINT, x0, y0, z0)
```

Makro

E1: Element Referenz {e1, e2, e3, ...}
POINT: reserviertes Wort
x0, y0, z0: Koordinaten des Punktes p0

Der Punkt bekommt die globalen Koordinaten (x0, y0, z0).

Beschreibung

Line

```
CREATE ELEMENT (E1, LINE, x1, y1, z1, x2, y2, z2)
```

Makro

E1: Element Referenz {e1, e2, e3, ...}
LINE: reserviertes Wort
x1, y1, z1; x2, y2, z2: die Koordinaten von zwei Punkten p1, p2

Die Linie ist in der Tat eine Strecke welche durch zwei Endpunkte definiert wird.

Beschreibung

Arc

Makro

```
CREATE ELEMENT (E1, ARC, x0,y0,z0, x1,y1,z1, x2,y2,z2, orient)
```

E1: Element Referenz {e1, e2, e3, ...}
ARC: reserviertes Wort
x0, y0, z0: Bogenzentrum p0
x1, y1, z1: Bogenanfangspunkt p1
x2, y2, z2: Bogenendpunkt p2
orient: orientation in contour (counterclockwise: 1; clockwise: -1). outer contour = 1; hole = -1. default value = +1.

Beschreibung

Der Bogen wird immer so interpretiert, dass er ausgehend vom Punkt p1 im Gegenuhrzeigersinn in Richtung von p2 gezeichnet wird, unabhängig davon, welcher der beiden Winkel (Winkel 1, Winkel 2) der kleinere ist. Anders ausgedrückt, der Benutzer muss die beiden Punkte p1 und p2 so setzen, dass diese Regel zutrifft. Der Radius des Kreisbogens wird als Länge der Strecke p0-p1 bestimmt. Der Endpunkt p2 wird nur noch zur Bestimmung des Endwinkels genutzt. Falls p2 nicht auf dem Bogenradius liegt, dann hat dies keinen Einfluss auf den Radius des Bogens. Die letzte Variable 'orient' kann die Werte {1, -1} annehmen und legt schliesslich fest in welcher Umlaufrichtung eine Kontour abgefahren wird. Im Normalfall und bei äusseren Konturen ist orient = +1.

Circle

Makro

```
CREATE ELEMENT (E1, CIRCLE, cx, cy, cz, nx, ny, nz, R)
```

E1: Element Referenz {e1, e2, e3, ...}
CIRCLE: reserviertes Wort
cx, cy, cz: Kreiszentrum p0
nx, ny, nz: homogener, Richtungsvektor senkrecht zur Kreisscheibe
R: Kreisradius

Beschreibung

Als Erstes erzeugen wir eine Ebene durch den Ursprung. Die Richtung dieser Ebene soll dabei durch einen Normalenvektor festgelegt werden, also einen Vektor der seinen Fusspunkt im Ursprung hat und senkrecht auf der Ebene steht. Dieser homogene Vektor wird durch die drei Koordinaten (nx, ny, nz) vorgegeben. Damit ist die Ausrichtung der Ebene im Raum gegeben. Als weiterer Schritt wird diese Ebene nun parallel verschoben, so dass sie durch den Punkt (cx, cy, cz) verläuft. Damit ist die Kreisebene und das Kreiszentrum im Raum vollständig definiert. Mit dem Kreisradius R ist letztlich auch der Kreis in dieser Ebene bestimmt.

Polygon (konvex)

Makro

```
CREATE ELEMENT (E1, POLYGON,  n)
DATA(x1,y1,z1, ..., xi,yi,zi)
...
DATA(xj,yj,zj, ..., xn,yn,zn)
```

E1: Element Referenz {e1, e2, e3, ...}
POLYGON: reserviertes Wort
n: ganze Zahl = Anzahl Punkte des Polygons
xi, yi, zi ... : Koordinaten der Polygonpunkte

Beschreibung

Die Daten werden in einer beliebigen Anzahl 'DATA'-Linien angefügt. Es gilt darauf zu achten, dass die Anzahl Punkte in den Datenlinien mit der

deklarierten Anzahl Punkte ‘n’ übereinstimmt. Im Weiteren müssen alle z-Koordinaten gleich Null sein. Das folgende Beispiel zeichnet ein Achteck.

Beispiel

```
CREATE ELEMENT (E1, POLYGON,  8)
DATA( 2, 1, 0,  1, 2, 0, -1, 2, 0)
DATA(-2, 1, 0, -2,-1, 0, -1,-2, 0)
DATA( 1,-2, 0,  2,-1, 0)
```

Polyline

Makro

```
CREATE ELEMENT (E1, POLYLINE,  n)
DATA(x1,y1,z1, ..., xn,yn,zn)
...
DATA(xj,yj,zj, ..., xn,yn,zn)
```

E1: Element Referenz {e1, e2, e3, ...}
POLYLINE: reserviertes Wort
n: ganze Zahl = Anzahl Punkte der Polyline
xi, yi, zi ... : Koordinaten der Polylinepunkte

Beschreibung

Die Daten werden in einer beliebigen Anzahl ‘DATA’-Linien angefügt. Es gilt darauf zu achten, dass die Anzahl Punkte in den Datenlinien mit der deklarierten Anzahl Punkte ‘n’ übereinstimmt. Im Weiteren müssen alle z-Koordinaten gleich Null sein.

Quadstrip

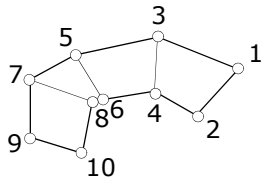
Makro

```
CREATE ELEMENT (E1, QUAD_STRIP, n)
DATA(x1,y1,z1, ..., xn,yn,zn)
```

E1: Element Referenz {e1, e2, e3, ...}
POLYLINE: reserviertes Wort
n: ganze Zahl = Anzahl Punkte der Polyline
xi, yi, zi ... : Koordinaten der Polylinepunkte

Beschreibung

Ein quadstrip ist wie der Name sagt ein Streifen bzw. eine Kette von Vierecken. Es ist wichtig die richtige Nummerierung der Punkte einzuhalten, so wie in nebenstehender Abbildung dargestellt. Die gesamte Punktzahl ist geradzahlig. Im Weiteren müssen alle z-Koordinaten gleich Null sein. Während ein Polygon nur konvexe Berandungen zulässt, sind mit einem ‘quadstrip’ auch konkave oder sogar aufgewickelte Strukturen möglich. Das nachfolgende Beispiel zeigt einen einfachen Winkelprofil-Querschnitt, welcher als Ausgangspunkt für eine Extrusion verwendet werden könnte.



Beispiel

```
CREATE ELEMENT (E1, QUAD_STRIP, 6)
DATA(0, 0, 0, 5, 0, 0, 5, 0.5, 0)
DATA(0.5, 0.5, 0, 0.5, 5, 0, 0, 5, 0)
```

Rohdaten Selektieren

Makro

```
SELECT ELEMENT (E1)
```

E1: Element Referenz {e1, e2, e3, ...}

Beschreibung

Für Mehrfachselektionen kann dieser Vorgang in einem Script mehrmals hintereinander für verschiedene Referenznummern aufgerufen werden. Die selektierten Elemente werden in den Ansichten entsprechend gekennzeichnet.

Rohdaten Löschen

Makro

```
CLEAR ELEMENT (E1 || SELECTION)
```

E1: Element Referenz {e1, e2, e3, ...}
SELECTION: (reserviertes Wort) -> some elements must be selected

Beschreibung

Wir verwenden hier in 'sonar script' ausdrücklich das Kommando '`CLEAR`' und nicht '`DELETE`' um Elemente zu Löschen. Das bedeutet, dass die gelöschten Elemente nicht in die Zwischenablage übertragen und folglich auch nicht mit '`PASTE`' wieder eingesetzt werden können. Das Löschen und wieder Einsetzen von Rohdaten innerhalb eines scripts macht letztlich keinen Sinn und ist nicht notwendig bzw. kann ggf. besser anders gelöst werden.

Makro

```
CLEAR ALL
```

Diese Anweisung löscht alle was selektiert ist, wie auch immer sich die Selektion zusammensetzt. Das können entweder nur Elemente oder eine Kombination aus Elementen und Objekten sein.

Makro

```
SET PROPERTY (E1, INUSE, bool)
```

E1: Element Referenz {e1, e2, e3, ...}
INUSE: reserviertes Wort
bool: bool'scher Wert {TRUE || FALSE)

Beschreibung

Ein Element kann damit vorübergehend ausgeschaltet werden, indem man die Funktion mit dem dritten Parameter 'FALSE' auf das referenzierte Element anwendet. Wird das Modell anschliessend irgendwann gesichert, dann ist das betreffende Element definitiv gelöscht. Andernfalls kann es zu einem späteren Zeitpunkt mit dem Parameter 'TRUE' wieder reaktiviert werden.

Rohdaten Gruppieren

Makro

```
GROUP ELEMENTS (SELECTION || ALL)
```

SELECTION: (reserviertes Wort) -> some selected elements must be grouped
ALL: (reserviertes Wort) -> all elements in the model must be grouped

Beschreibung

Der Gruppierungsprozess versucht entweder die selektierten oder alle Elemente so weit als möglich zu gruppieren. Dies bedeutet nicht, dass die betreffenden Elemente alle zu einer einzigen Gruppe zusammengefasst werden. Elemente welche nahtlos aneinander passen werden in gleiche Gruppen aufgenommen. Es wird versucht geschlossene Ketten von Elementen zu bilden. Wo das nicht gelingt bleiben offene Ketten übrig. Die Funktion fügt also lediglich zu Gruppen zusammen was zueinander passt. Dabei werden ggf. mehrere oder sehr viele Gruppen gebildet, falls die räumliche Anordnung der Elemente dies ergibt. Einzelne Elemente für die keine Anschlusspartner gefunden werden, bleiben bei diesem ganzen Vorgang frei bzw. werden nicht an Gruppen angefügt.
Beim Gruppierungsprozess spielt die Gruppierungstoleranz eine Rolle. Dies ist ein Wert der darüber entscheidet, wie weit weg ein Anschlusspunkt eines

Elementes maximal sein darf, damit eine Verbindung der beiden Elemente noch zustande kommt. Dieser Toleranzwert kann in den Präferenzen des Prgramms eingestellt werden.

```
UNGROUP ELEMENTS (SELECTION || ALL)
```

Makro

SELECTION: (reserviertes Wort) -> some selected elements must be grouped
ALL: (reserviertes Wort) -> all elements in the model must be grouped

Beschreibung

UNGROUP ist die Umkehrfunktion von GROUP und löst die Gruppierung wieder auf. Die betreffenden Elemente gehören anschliessend keiner Gruppe mehr an und sind wieder frei.
Statt eine bereits bestehende Anzahl von Elementen zu gruppieren, können diese auch importiert werden mit folgender Anweisung:

Rohdaten für ein Zahn- oder Kettenrad

```
CONCATENATE ELEMENTS (E1, RING, x0, y0, n)
```

Makro

E1 : Element Referenz {e1, e2, e3, ...} -> E1: Repräsentant eine Elementgruppe
RING : reserviertes Wort
x0, y0 : Bogen- oder Kreiszentrum p0 in der X-Y-Ebene (Front View)
n : Anzahl der Wiederholungen längs dem Bogen oder Kreis

Beschreibung

Die Anweisung ist eine spezielle Funktion, welche sich wiederholende Untergruppen von Elementen zu einer grösseren Gruppe zusammenschliesst. Die Funktion wurde ursprünglich dazu geschaffen, einen einzelnen Zahn eines Zahnrades zu einem ganzen Zahnrad zu erweitern. Die Funktion erwartet als Input eine Gruppe von Elementen, ein Drehzentrum und die Zähnezahl. Mit diesen Informationen trägt die Funktion das gegebene 'Zahnbild' bzw. die Elementgruppe längs einem Kreis gleichverteilt so oft ab, wie die Zähnezahl angibt. Es liegt in der Verantwortung des Benutzers, das Zahnbild bzw. die einzelnen Elemente geometrisch so auszulegen, dass diese nahtlos aneinander passen, wenn die Gruppe dupliziert und um das Drehzentrum gedreht wird.

Rohdaten in Konturen umwandeln

```
CREATE CONTOUR_LINE_ARC(C1, E1 || SELECTION)
```

Makro

C1: Contour Referenz {c1, c2, c3, ...}
E1: Element Referenz {e1, e2, e3, ...}, Mitglied einer geschlossenen Gruppe von Elementen
SELECTION: (reserviertes Wort) -> eine geschlossene Gruppe von selektierten Elementen

Beschreibung

Eine referenzierte Gruppe oder eine Selektion von Elementen wird in eine Kontur umgewandelt. Eine Kontur ist eine geschlossene Gruppe von Elementen. Eine Kontur unterscheidet sich von einer Gruppe von Elementen dadurch, dass die Elemente einer Kontur in einem gewissen Sinne ihre Eigenständigkeit aufgeben und zusammen eine neue Datenstruktur bilden. Die Kontur bekommt im sog. Objekt-Tool einen neuen Eintrag unter dem Ordner 'Contour' während die Elemente aus dem 'Elementen'-Ordner, wo sie vorher eingetragen waren, verschwinden. Eine Kontur kann anschliessend extrudiert werden

Konturen können als Einheit selektiert werden mit

```
SELECT CONTOUR (C1)
```

Makro

C1: Contour Referent {c1, c2, c3, ...} die vorher gesetzt wurde

Rohdaten in eine Polyline umwandeln

Makro

```
TRANSFORM ELEMENTS (E1, LINE_SEGMENTS, dL)
```

E1: Element Referenz {e1, e2, e3, ...} -> Mitglied einer geschlossenen Gruppe von Elementen
LINE_SEGMENTS: reserviertes Wort
dL: Länge der zu erzeugenden Streckenabschnitte der Polyline

Beschreibung

Die Funktion war ursprünglich dazu gedacht, einen geschlossenen Pfad bestehend aus Linien und Bogen in eine Polyline mit konstanten Streckenlängen zu unterteilen um das Resultat als Basis für eine Kette zu verwenden. Die Funktion kann natürlich im Rahmen dieser Funktionalität auch für andere Aufgaben verwendet werden.
Die Funktion funktioniert sinngemäss ähnlich wie wenn der Navigator mit dem Zirkel längs einer Kurve x-mal das Zirkelmass abträgt, um herauszufinden wie lange es dauert bis er an einem bestimmten Zielpunkt ist (diese Analogie ist für Leute die sich mit den Methoden vor dem Computerzeitalter noch auskennen).

Rohdaten bewegen

Makro

```
MOVE ELEMENT (E1 || SELECTION, MOVE_MATRIX, O1 || SELECTION)
```

E1: Element Referenz {e1, e2, e3, ...} oder eine Selektion von Elementen
MOVE_MATRIX : reserviertes Wort
O1 : eine Objekt Referenz oder ein selektiertes Objekt

Beschreibung

Die Aufgabe dieser Anweisung besteht darin, ein Element oder eine Gruppe von Elementen so im Raum zu drehen, wie es von der Drehmatrix eines bereits vorhandenen und gedrehten Objektes vorgegeben wird. Die Funktion dreht folglich das Element E1 mit der Matrix von Objekt O1.

Bemerkung

Elemente werden in sonar kaum bewegt. In der Regel werden die Rohdaten dort gezeichnet wo in der Folge die Objekte entstehen sollen, d.h. in der Nullstellung der zukünftigen Objekte. Es sind schliesslich die Objekte selbst die im Raum gedreht und verschoben werden, nachdem sie erzeugt wurden.

Rohdaten Importieren

Makro

```
IMPORT COLLECTION_LINE_ARC (E1, FILENAME, "filename")
```

E1: Element Referenz {e1, e2, e3, ...}
FILENAME: reserviertes Wort
"filename": Ein regulärer Filename zwischen Anführungs- und Schlusszeichen.

Beschreibung

Das Resultat dieser Operation ist eine Gruppe von Elementen mit einer neuen Gruppennummer. Die Referenz 'E1' zeigt dabei auf ein Element der Gruppe. Die importierten Elemente sind noch nicht zu einer Kontur zusammengefasst, so wie das im nächsten Abschnitt vorgeführt wird. Es handelt sich einfach um eine Anzahl von Elementen mit einer gemeinsamen Gruppennummer.

Makro

```
IMPORT CONTOUR_LINE_ARC (C1, FILENAME, "filename")
```

C1: Contour Referent {c1, c2, c3, ...}
FILENAME: reserviertes Wort
"filename": Ein regulärer Filename zwischen Anführungs- und Schlusszeichen.

Der Unterschied dieser Anweisung zur vorangehenden besteht darin, dass letztere am Ende vom Typ 'Contour' ist. während im ersten Fall lediglich eine Gruppe von Elementen vorliegt. Eine 'Contour' ist eine Weiterverarbeitung von Elementen mit zusätzlichen Merkmalen und einem eigenen Eintrag im 'Object Tool'.

Beschreibung

```
IMPORT POLYLINE (E1, FILENAME, "filename")
```

syntax

E1: zugewiesene Element Referenz {e1, e2, e3, ...}
FILENAME : reserviertes Wort
"filename": vollständiger Dateipfad des Files. Gänsefüsschen müssen geschrieben werden.

Eine Polyline bzw. eine durch Strecken verbundene Punktfolge wird geladen. Die maximale Anzahl Punkte beträgt 4096. Das neue Element bekommt die Elementreferenz wie im ersten Parameter angegeben.

Beschreibung

Rohdaten eine Orientierung geben

Bei der Weiterverarbeitung der Rohdaten in Objekte ist bei gewissen Operationen nicht immer klar welche Seite die Innen- und welche die Aussenseite ist. So gibt es z.B. Funktionen welche Gruppen von Elementen zu Rotationsflächen umwandeln oder ähnliche Funktionen welche auch nicht geschlossene Konturen, also einzelne Linien-Bogen-Gruppen, akzeptieren. In diesen Fällen muss den betreffenden Funktionen bekannt gemacht werden, welche Seite der Fläche innen und welche aussen ist. Der Benutzer hat die Möglichkeit diese Information bereits in den einzelnen Elementen zu speichern, indem er diesen einen Normalenvektor zuordnet.

Beschreibung

```
SET PROPERTY (E1, NORMALVECTOR, x1, y1, z1)  -- LINE
SET PROPERTY (E1, NORMALDIRECTION, ±1)       -- ARC
```

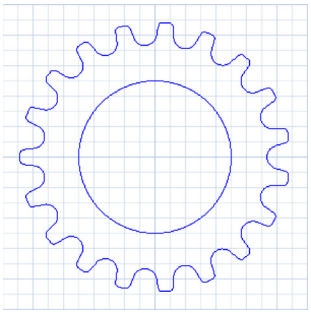
Syntax

E1: Element Referenz {e1, e2, e3, ...}
NORMALVECTOR, NORMALDIRECTION: reservierte Worte
x1, y1, z1: Ein homogener Vektor welcher senkrecht auf einer Linie steht und nach 'Aussen' zeigt.
±1: Richtungsangabe der Aussenseite des Bogens: +1: zeigt vom Bogenzentrum Richtung Bogen.
-1: zeigt vom Bogen Richtung Bogenzentrum.

Beispiel

Makro

```
IMPORT COLLECTION_LINE_ARC (E1, FILENAME, "C:\sonar\gear.txt")
CONCATENATE ELEMENTS (E1, RING, 0, 0, 19)
UNGROUP ELEMENTS (ALL)
GROUP ELEMENTS (ALL)
CREATE ELEMENT (E2, CIRCLE, 0, 0, 0, 0, 0, 1.0, 1.25)
TRANSFORM ELEMENTS (E1, CONTOUR)
```



Das script importiert die Form eines einzelnen Zahn's bestehend aus Linien und Bogen. Die zweite Anweisung des scripts dupliziert diesen Zahn 19 mal um das Zentrum herum. Anschliessend bewirkt die dritte Linie, dass die ursprüngliche Gruppen-Eigenschaft des Zahnbildes, welche nun 19 mal dupliziert wurde, aufgelöst wird. Beim Duplizieren bekam jede Zahnkopie eine neue eigene Gruppennummer. Die dritte Linie hat also zur Folge, dass sämtliche Elemente nun ungruppiert sind. Mit der vierten Linie gruppieren wir schliesslich alle Elemente zu einer einzigen neuen Gruppe. Anschliessend wird noch der Innenkreis erzeugt und zum Schluss wird die ganze Gruppe in eine Contour umgewandelt.

Primitivkörper

Einführung

Primitivkörper sind die physikalischen Grundbausteine in ‘sonar’. Viele Primitivkörper können auf direktem Weg mit einer Anweisung erzeugt werden. Andere nehmen den Weg über die Rohdaten indem Rohdatenkonturen extrudiert werden. Aus Primitivkörpern können später durch Zusammensetzen und Verlinken komplexere Objekte erzeugt werden. In sonar gibt es auch Funktionen die ihrerseits direkt komplexere Objekte aus vielen Primitivkörpern erzeugen. Eine Zugfeder ist ein solches Beispiel.

Im Folgenden wird die Erzeugung jedes Primitivkörpers im Einzelnen besprochen. Für gewisse Primitivkörper gibt es mehrere unterschiedliche Anweisungen. Anweisungen, welche ein Objekt in einer einzigen Anweisung auch gleich in eine gedrehte Raumlage bringen, sind oft nicht für jedermann verständlich. Es ist manchmal nicht ganz trivial drei Drehungen um drei verschiedene Achsen im Kopf zu addieren. Makros mit einer grossen Anzahl solcher Drehungen sind schlecht leserlich. Aus diesem Grund bevorzugen viele Benutzer mit Recht die Durchführung von mehreren einfacheren Operationen von der Art:

1. Den Körper zuerst in der Nulllage definieren (ohne Drehungen und ohne Translationen)
2. Den Körper mit mehreren einzelnen Drehungen um jeweils eine Achse in die richtige Drehlage bringen.
3. Den Körper an die Raumposition verschieben (Translation)

Natürlich kann man jede Makro-Anweisung auch dazu benutzen einen Körper in der unverdrehten Nulllage zu erzeugen, indem man die betreffenden Parameter welche Drehungen und Translationen spezifizieren einfach auf Null setzt.

Sphere (Kugel)

```
CREATE OBJECT (o1, SPHERE, x0, y0, z0, R)
```

Makro

o1: Objekt Referenz {o1, o2, o3, ...}
SPHERE: reserviertes Wort
x0, y0, z0: Koordinaten des Kugelzentrums
R: Radius der Kugel

Beschreibung

Die erzeugte Kugel befindet sich mit ihrem Kugelzentrum an den Koordinaten x0, y0, z0. Eine Kugel ist in sonar eine analytisch exakte Kugeloberfläche, d.h. die Kugel ist nicht segmentiert und wird nicht durch Flächen angenähert. Eine rollende Kugel scheppert nicht in ‘sonar’.

```
CREATE OBJECT (o1, PARTICLE_SPHERE, x0, y0, z0, R)
```

Makro 2
(im Basismodul ev. nicht vorhanden)

o1: Objekt Referenz {o1, o2, o3, ...}
PARTICLE_SPHERE: reserviertes Wort
x0, y0, z0: Koordinaten des Kugelzentrums
R: Radius der Kugel bzw. des Partikels

Cylinder (Zylinder)

Makro 1

```
CREATE OBJECT (o1, CYLINDER, x0, y0, z0, wx, wy, wz, R, dz)
```

o1: Objekt Referenz {o1, o2, o3, ...}
CYLINDER: reserviertes Wort
x0, y0, z0: Koordinaten des Zylinderzentrums
wx, wy, wz: Drehwinkel relativ zu den drei Koordinatenachsen (Drehungen um das Zylinderzentrum)
R: Zylinderradius
dz: Zylinderlänge

Beschreibung

1. Erzeugung eines Zylinders mit der Zylinderachse auf der z-Achse. Der Schwerpunkt des Zylinders befindet sich im Ursprung.
2. Ausgehend von dieser Raumlage wird der Zylinder um die gegebenen Winkel um die drei Koordinatenachsen gedreht, und zwar in der Reihenfolge x, y, z. Der Schwerpunkt des Zylinders befindet sich anschliessend immer noch im Ursprung
3. Der Zylinder wird unter Beibehaltung seiner aktuellen Drehlage translatorisch an seinen Bestimmungsort x0, y0, z0 verschoben.

Makro 2

```
CREATE OBJECT (o1, CYLINDER, x1, y1, z1, x2, y2, z2, R)
```

o1: Objekt Referenz {o1, o2, o3, ...}
CYLINDER: reserviertes Wort
x1, y1, z1 ; x2, y2, z2: Koordinaten der beiden Achsenendpunkte des Zylinders
R: Zylinderradius

Beschreibung

1. Die beiden Punkte (x1, y1, z1) und (x2, y2, z2) definieren die Zylinderachse. Es wird ein Zylinder mit dem Radius R um die gegebene Achse gelegt. Die beiden kreisförmigen Endscheiben des Zylinders haben ihre Kreiszentren in den beiden gegebenen Punkten und stehen senkrecht auf der Achse.

Zylinder mit abgerundeten Kanten

Makro

Die Erzeugung eines Zylinders mit abgerundeten oder angeschrägten Kanten (cylinder bevelled) geschieht durch Abändern eines normalen Zylinders bzw. durch Setzen einer entspr. Eigenschaft für den betr. Zylinder.

```
SET PROPERTY (o1, BEVEL, {ROUND || FACETTE}, r)
```

o1: Objekt Referenz auf einen existierenden Zylinder {o1, o2, o3, ...}
BEVEL, ROUND, FACETTE: reservierte Wörter
r: Rundungsradius an der Kante bzw. Facettenbreite (r*45°)

Beschreibung

Das Setzen dieser Eigenschaft ändert die scharfen Kanten eines Zylinders. Es sind abgerundete oder angeschrägte Kanten möglich. Dazu setzt man den gewünschten parameter `ROUND` oder `FACETTE`. Die angeschrägten Kanten verstehen sich als Anschrägung um die Breite 'r' und um 45°.

Kegelstumpf, Kegel

Makro 1

```
CREATE OBJECT (o1, CONE, x0, y0, z0, wx, wy, wz, R, r, dz)
```

o1: Objekt Referenz {o1, o2, o3, ...}
CONE: reserviertes Wort
x0, y0, z0: Koordinaten des Zentrums. Das Zentrum ist die halbe Kegelstumpfhöhe.
wx, wy, wz: Drehwinkel relativ zu den drei Koordinatenachsen (Drehungen um das Zentrum)

R: grosser Kegelstumpfradius
r: kleiner Kegelstumpfradius
dz: Kegelstumpfhöhe

Beschreibung

Einen Kegelstumpf kann man sich als einen veränderten Zylinder vorstellen, bei dem der eine Endkreisradius verkleinert wird. Ein Kegel ist ein Spezialfall des Kegelstumpfes mit r = 0.

1. Erzeugung eines Kegelstumpfes mit der Kegelachse auf der z-Achse. Der Kegel zeigt mit seiner Spitze in Richtung positive z-Achse. Das Zentrum des Kegelstumpfes befindet sich im Ursprung. Das Zentrum ist allerdings nicht mit dem Schwerpunkt identisch, sondern befindet sich am Ort der halben Kegelstumpfhöhe.
2. Ausgehend von dieser Raumlage wird der Kegelstumpf um die gegebenen Winkel um die drei Koordinatenachsen gedreht, und zwar in der Reihenfolge x, y, z. Das Zentrum des Kegelstumpfes befindet sich anschliessend immer noch im Ursprung
3. Der Kegelstumpf wird unter Beibehaltung seiner aktuellen Drehlage translatorisch an seinen Bestimmungsort x0, y0, z0 verschoben.

Makro 2

```
CREATE OBJECT (o1, CONE, x1, y1, z1, x2, y2, z2, R, r)
```

o1: Objekt Referenz {o1, o2, o3, ...}
CONE: reserviertes Wort
x1, y1, z1 ; x2, y2, z2: Koordinaten der beiden Achsenendpunkte des Kegelstumpfes
R: grosser Kegelstumpfradius in Punkt (x1, y1, z1)
r: kleiner Kegelstumpfradius in Punkt (x2, y2, z2)

Beschreibung

1. Die beiden Punkte (x1, y1, z1) und (x2, y2, z2) definieren die Kegelachse. Es wird ein Kegelstumpf mit den Radien R und r um die gegebene Achse gelegt. Die beiden kreisförmigen Endscheiben des Kegelstumpfes haben ihre Kreiszentren in den beiden gegebenen Punkten und stehen senkrecht auf der Achse.

Rohr

Makro

```
CREATE OBJECT (o1, TUBE, x0, y0, z0, wx, wy, wz, R, r, dz)
```

o1: Objekt Referenz {o1, o2, o3, ...}
TUBE: reserviertes Wort
x0, y0, z0: Koordinaten des Rohrzentrums
wx, wy, wz: Drehwinkel relativ zu den drei Koordinatenachsen (Drehungen um das Rohrzentrum)
R: Rohr-Aussendurchmesser
r: Rohr-Innendurchmesser
dz: Rohrlänge

Beschreibung

1. Erzeugung eines zylindrischen Rohrs mit der Rohrachse auf der z-Achse. Der Schwerpunkt des Rohrs befindet sich im Ursprung.
2. Ausgehend von dieser Raumlage wird das Rohr um die gegebenen Winkel um die drei Koordinatenachsen gedreht, und zwar in der Reihenfolge x, y, z. Der Schwerpunkt des Rohrs befindet sich anschliessend immer noch im Ursprung
3. Das Rohr wird unter Beibehaltung seiner aktuellen Drehlage translatorisch an seinen Bestimmungsort x0, y0, z0 verschoben.

Rohr Segment

Makro

```
CREATE OBJECT (o1, TUBE_SEGMENT, x1,y1,z1, x2,y2,z2, R,r,phi,ws)
```

o1: Objekt Referenz {o1, o2, o3, ...}
TUBE_SEGMENT: reserviertes Wort
x1, y1, z1 ; x2, y2, z2: Koordinaten der beiden Achsenendpunkte der Rohrachse
R: Rohr-Aussendurchmesser
r: Rohr-Innendurchmesser
phi: Azimutale Ausrichtung des Segmentmittelpunktes um die z-Achse [°]
ws: abs. Winkel des Rohrsegmentes längs dem Umfang (Segmentbreite) [°]

Beschreibung

Man kann sich die Entstehung eines Rohrsegmentes wie folgt vorstellen: Wir gehen von einem ganzen Rohr aus, wie es die entspr. Funktion erzeugt (TUBE). Das Rohr ist längs der z-Achse ausgerichtet. Zuerst schneiden wir aus dem Rohr in Längsrichtung einen Streifen heraus, welcher zur Rohrachse einen Bogenwinkel = ws bildet. Der Rest des Rohres wird entfernt. Anschliessend wird das verbliebene Segment um die Rohrachse bzw. die z-Achse um den Winkel (phi) in die gewünschte Azimutlage gedreht. Das resultierende Rohr ist immer noch unendlich lang, denn wir haben bislang noch keine Längenangaben gemacht. Die Funktion beinhaltet die Angabe von zwei Koordinaten (`x1,y1,z1, x2,y2,z2)`, durch welche die Achse des Rohres bzw. des daraus herausgeschnittenen Segmentes gehen soll. Mit zwei Drehungen um die x- resp. y-Achse und einer Translation wird die Rohrachse mitsamt Segment nun in die gewünschte Raumlage gedreht. Am Ende wird das Segment senkrecht zu seiner Achse durch die beiden Endpunkte abgeschnitten.

Rohr Oberfläche (Tube Surface)

Makro

```
CREATE OBJECT (O1, TUBE_SURFACE, x0, y0, z0, wx, wy, wz, R, dz
```

o1: Objekt Referenz {o1, o2, o3, ...}
TUBE_SURFACE: reserviertes Wort
x0, y0, z0 ;
wx, wy, wz:
R: Rohr-Aussendurchmesser
dz: Rohrlänge

Beschreibung

Eine Rohrfläche kann eine äussere begrenzende zylindrische Fläche sein welche den Simulationsbereich begrenz. Man könnte dies zwar meistens auch mit einem Rohr bewerkstelligen. Eine einfache Zylinderfläche ist aber vom Rechenaufwand her gesehen einfacher und schneller. In dieser Beziehung ist die Rohroberfläche verwandt mit der Ebene (PLANE) welche dasselbe mit einer flachen Oberfläche macht.

Cuboid (Quader)

Makro 1

```
CREATE OBJECT (o1, CUBOID, x1, y1, z1, x2, y2, z2)
```

o1: Objekt Referenz {o1, o2, o3, ...}
CUBOID: reserviertes Wort
x1, y1, z1 ; x2, y2, z2: Koordinaten von 2 diagonal gegenüberliegenden Ecken in der Nullage

Beschreibung

Mit dieser Anweisung wird ein Quader in einer unverdrehten Raumlage erzeugt. Die Kanten des Quaders sind alle parallel zu den Koordinatenachsen. Die beiden Punkte (`x1,y1,z1)` und `(x2,y2,z2)` definieren zwei diagonal gegenüberliegende Eckpunkte des Quaders. Setzt man die Koordinaten x2 = -x1, y2 = -y1, z2 = -z1, dann erhält man einen Quader in der Nulllage.

Makro 2

```
CREATE OBJECT (o1, CUBOID, x0, y0, z0, wx, wy, wz, dx, dy, dz)
```

o1: Objekt Referenz {o1, o2, o3, ...}

CUBOID: reserviertes Wort
x0, y0, z0: Koordinaten des Quaderzentrums
wx, wy, wz: Drehwinkel relativ zu den Koordinatenachsen (Drehungen um das Quaderzentrum)
dx, dy, dz: Kantenlängen des Quaders in der Nullage in Richtung der drei Koordinatenachsen

Beschreibung

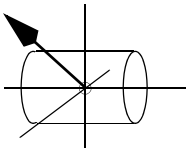
Diese Anweisung definiert einen Quader direkt in einer verdrehten Raumlage. Zuerst wird ein Quader mit den Kantenlängen (`dx,dy,dz`) mit seinem Schwerpunkt im Ursprung erzeugt (Nulllage). Anschliessend wird der Quader mit den Winkeln (`wx,wy,wz`) um die drei Koordinatenachsen gedreht. Dabei gilt es zu beachten, dass die Drehfolge um die drei Achsen gegeben ist. Schliesslich findet eine Translation mit dem Vektor (`x0,y0,z0`) in die endgültige Raumlage statt.

Torus

Makro

```
CREATE OBJECT (o1, TORUS, R, r)
```

o1: Objekt Referenz {o1, o2, o3, ...}
TORUS: reserviertes Wort
R: mittlerer Ringradius des Torus
r: Radius des kreisförmigen Querschnitts des Ringes



Beschreibung

Der Torus befindet sich, so wie er erzeugt wird, in seiner Nulllage. Die Torusachse, d.h. die Richtung mit der man den Finger durch den Ring stecken würde, ist identisch mit der z-Achse. Der Ringkreis mit dem Radius R befindet sich in der x-y-Ebene des Koordinatensystems und das Zentrum dieses Kreises ist mit dem Ursprung identisch. Das Programm erwartet, dass R > 2r, d.h. dass der Torus tatsächlich ein Loch hat.

Torus Segment

Makro

```
CREATE OBJECT (o1, TORUS_SEGMENT, R, r, phi)
```

o1: Objekt Referenz {o1, o2, o3, ...}
TORUS_SEGMENT: reserviertes Wort
R: mittlerer Ringradius des Torus
r: Radius des kreisförmigen Querschnitts des Ringes
phi: Bogenwinkel des Segmentes [°]

Beschreibung

Die räumliche Lage des Torus-Segments entspricht derjenigen eines ganzen Torus, aus dem ein Kuchenstück herausgeschnitten wird. Das Segment befindet sich in seiner Nullage ausgemittet zur y-Achse, d.h. der Schwerpunkt des Segments liegt auf der y-Achse und die beiden seitlichen Begrenzungsflächen des Torus liegen je phi/2 links und rechts von der y-Achse.

Anwendung

Ein Beispiel für eine Anwendung von Torus Segmenten ist die Zug- oder Druckfeder, welche stückweise aus solchen Objekten zusammengesetzt wird.

Prisma (konvex)

Makro

```
CREATE OBJECT (o1, PRISM, E1, EXTRUSION, dz) -- convex only
```

o1: Objekt Referenz {o1, o2, o3, ...}
PRISM, EXTRUSION: reserviertes Wort
E1: Element Referenz auf ein vorher erzeugtes konvexes Polygon
dz: Prismalänge (extrudierte Länge)

Beschreibung

Für nicht-konvexe Prismen verweisen wir auf den Primitivkörper vom Typ ‘PRISM_QUAD_STRIP’. Der Definition eines Prismas geht die Erzeugung eines (konvexen) Polygons voraus. Die Element-Definition hinterlässt eine Referenz auf das betreffende Polygon, welche wir hier bei der Erzeugung des Prismas einsetzen. Das Polygon muss in der x-y-Ebene liegen. Die Extrudier-Richtung ist die z-Achse. Das erzeugte Prisma liegt mittig vor und hinter der x-y-Ebene. Der Schwerpunkt liegt in der x-y-Ebene.

Beispiel

```
CREATE ELEMENT (E8, POLYGON,  n)
DATA(x1,y1,z1, ..., xi,yi,zi)
...
DATA(xj,yj,zj, ..., xn,yn,zn)
CREATE OBJECT (o1, PRISM, E8, EXTRUSION, dz)
```

Im Allgemeinen werden die Punkte des Polygons übersichtshalber in mehreren ‘DATA’-Linien angeschrieben (z.B. 3 Punkte pro DATA-Anweisung).

Prisma (quadstrip)

Makro

```
CREATE OBJECT (o1, PRISM_QUAD_STRIP, E1, EXTRUSION, dz)
```

o1: Objekt Referenz {o1, o2, o3, ...}
PRISM_QUAD_STRIP, EXTRUSION: reserviertes Wort
E1: Element-Referenz auf ein existierendes Polygon, welches sich als ‘quadstrip’ darstellen liess.
dz: Prismalänge (extrudierte Länge)

Beschreibung

Ein ‘quadstrip’ ist ein Polygon bestehend aus einer Kette von aneinander gereihten, planen Vierecken, welche zusammen eine gemeinsame, im allg. nicht konvexe, äussere Hülle bilden. Der Name ‘quadstrip’ ist der openGL-Grafik Software entnommen, welche diese Formelemente dazu benutzt um durch Kombinationen komplexere Körper zu generieren. Nach dem konvexen Polygon ist der ‘quadstrip’ die nächste Ausbaustufe um kompliziertere Berandungen zu erzeugen. Auch der ‘quadstrip’ ist aber letztlich ein Polygon und besteht folglich ausschliesslich aus geraden Streckenelementen längs seiner Berandung. Radien können ggf. mit mehreren kleinen Streckenelemente angenähert werden.

Das ‘quadstrip’-Polygon muss in der x-y-Ebene liegen. Die Extrudier-Richtung ist die z-Achse. Das erzeugte Prisma liegt mittig vor und hinter der x-y-Ebene. Der Schwerpunkt liegt in der x-y-Ebene.

Prisma (Line-Arc)

Makro

```
CREATE OBJECT(o1, PRISM_LINE_ARC, E1 || SELECTION, EXTRUSION, dz)
```

o1: Objekt Referenz {o1, o2, o3, ...}
PRISM_LINE_ARC, SELECTION, EXTRUSION: reservierte Wörter
E1: Element-Referenz auf ein existierendes Element (LINE || ARC).
SELECTION: statt ein referenziertes Element soll ein selektiertes Element benutzt werden.

Beschreibung 1

Das referenzierte oder selektierte Element muss Mitglied einer Elementgruppe sein, welche zusammen eine geschlossene Kontur bildet. Alle Elemente müssen in der x-y-Ebene liegen. Die Kontur darf eine beliebige Kombination von Linien und Bogen haben. Eine typische Anwendung ist z.B. die Aussenkontur eines Zahnrades. Die Extrudier-Richtung ist die z-Achse. Das erzeugte Prisma liegt mittig vor und hinter der x-y-Ebene. Der Schwerpunkt liegt in der x-y-Ebene.

Eine Besonderheit dieser Funktion ist die Möglichkeit in einem Zug auch noch Löcher in der Kontur zu definieren. Das folgende Beispiel definiert eine Lasche einer Zahnkette mit 2 Löchern wie folgt:

Beispiel

```
IMPORT CONTOUR_LINE_ARC(E1, FILENAME, "C:\Import\Lasche.txt")
CREATE ELEMENT(E2, CIRCLE, 0, 0, 0, 0, 0, 1.0, 0.1605)
CREATE ELEMENT(E3, CIRCLE, 0.792, 0, 0, 0, 0, 1.0, 0.1605)
DESELECT ALL
SELECT ELEMENT(E1)
SELECT ELEMENT(E2)
SELECT ELEMENT(E3)
CREATE OBJECT(O1, PRISM_LINE_ARC, SELECTION, EXTRUSION, 0.12)
```

FIGURE 3. Lasche einer Maschinen- oder Zahnkette

Die erste Anweisung lädt eine bereits vorhandene 'Line-Arc'-Kontur welche als Datei mit dem Namen Lasche.txt im Verzeichnis 'C:\Import' bereit steht. In zwei weiteren Anweisungen wird diese Kontur mit zwei Kreisen ergänzt, welche explizit gezeichnet werden. Anschliessend werden alle drei Elemente selektiert und die Funktion zur Erzeugung des Prismas aufgerufen. Das Resultat ist eine Lasche mit Löchern.

Beschreibung 2

Die Funktion zur Erzeugung des Prismas (Prism Line-Arc) funktioniert also so, dass der Funktion vorerst ein Verweis auf eine Kontur übergeben wird. Diese erste Kontur ist immer die Aussenkontur des zu erzeugenden Prismas. Wenn am Ende der Kontur weitere selektierte Elemente folgen, dann werden diese immer als Löcher interpretiert, welche in der Kontur enthalten sein sollten. Diese zusätzlichen Elemente sollten immer vom Typ 'CIRCLE' sein. Grundsätzlich darf eine beliebige Anzahl von Kreisen folgen.

Twisted Prism (verdrehtes Prisma)

Makro

```
TWIST OBJECT (o1 || SELECTION, Tx, Ty, Tz, dz)
```

o1: Objekt Referenz {o1, o2, o3, ...}
SELECTION: reserviertes Wort. Statt ein referenziertes Objekt soll ein selektiertes Objekt benutzt werden.
Tx, Ty, Tz: Koordinaten eines Punktes im kartesischen Koordinatensystem
dz: Prismalänge (extrudierte Länge)

Beschreibung

Die Funktion benutzt ein existierendes konvexes Prisma und verdreht dieses nach Massgabe eines sog. Verdreh-Punktes. Der Verdrehpunkt (Tx,Ty,Tz) ist ein Punkt im kartesischen Koordinatensystem, welcher wie folgt benutzt wird. Die Funktion geht von der Nullposition des Prismas aus, dessen Schwerpunkt im Ursprung liegt und dessen Frontseiten parallel zur x-y-Ebene verlaufen. Das Prisma wurde bekanntlich längs der z-Achse extrudiert. Die Funktion versucht nun das Prisma um die y-Achse zu verdrehen. Dazu wird eine parallele Gerade zur x-Achse genommen und längs der y-Achse bis zur Koordinate Ty verschoben. Anschliessend wird diese Gerade um die y-Achse gedreht bis sie durch den gegebenen Punkt verläuft (Tx,Ty,Tz). Die x-Achse und die verdrehte Gerade bilden so gesehen ein Geradenpaar welches eine verdrehte Ebene im Raum festlegt.

Am besten stellt man sich einen ebenen Papierstreifen längs der y-Achse vor, welcher an seinem unteren Ende festgehalten und an seinem oberen Ende um die y-Achse verdreht wird. Die Verdrehung des Papierstreifens erfolgt dabei gleichmässig längs der y-Achse. Das Resultat ist ein Propeller-ähnlicher Papierstreifen.

Plane (Ebene)

Makro

```
CREATE OBJECT (O1, PLANE, nx0, ny0, nz0, nx1, ny1, nz1)
```

o1: Objekt Referenz {o1, o2, o3, ...}
PLANE: reserviertes Wort
nx0, ny0, nz0, nx1, ny1, nz1: Fuss- und Kopf-Koordinate eines Normalenvektors auf der Ebene

Beschreibung

Eine Ebene ist eine 2-dimensionale Struktur, welche den Modellbereich einschränkt. Physikalisch wirkt eine Ebene wie eine Oberfläche eines 3D-Körpers. Man kann sich eine Ebene deshalb auch als eine Seite eines unendlich grossen Quaders vorstellen. Die Ebene wird durch einen Normalenvektor definiert, welcher mit seinem Fusspunkt senkrecht auf der Ebene steht. Der Vektor zeigt dabei in Richtung des benutzten Modellraums. Mit 6 Ebenen könnte man z.B. einen quaderförmigen Hohlraum schaffen, welcher ein Modell in alle Richtungen einschränkt.

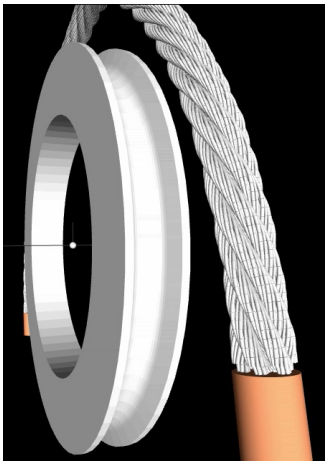
Rotational (Rotationskörper)

Makro

```
REVOLVE CONTOUR (O1, C1 || SELECTION)
```

o1: Objekt Referenz {o1, o2, o3, ...}
c1: Kontur Referenz
SELECTION: reserviertes Wort. Es soll ein selektiertes Element benutzt werden.

Beschreibung



Die Funktion setzt voraus , dass vorab eine geschlossene Kontur (Line-Arc-Contour) bereitgestellt wurde, welche den Querschnitt des zu erzeugenden Rotationskörpers definiert. Dieser Profil-Querschnitt muss in der X-Y-Ebene definiert sein, d.h. die z-Koordinaten der Punkte der Kontur sollen Null sein (wie immer bei Rohdaten). Die Kontur wird so gezeichnet, als ob man sie anschliessend um die X-Achse rotieren würde. Tatsächlich verschiebt das System bei der Ausführung der Funktion die Kontur im Hintergrund automatisch in die Z-Y-Ebene und rotiert sie um die Z-Achse. Es entsteht ein Rotationskörper mit der Z-Achse als Rotationsachse.

Der Name 'Kontur' impliziert, dass es sich um eine geschlossene Kontur handelt und die gruppierten Elemente (Linien, Bogen) bereits in eine 'contour' umgewandelt wurden. Die Kontur wurde selektiert bevor die Funktion `'REVOLVE'` aufgerufen wird. Zusammen besitzt die Funktion dann genug Informationen, um die selektierte Kontur (Querschnitt) um die Drehachse zu rotieren und zu einem Rotationskörper zu verarbeiten.

FIGURE 4. eine typische Anwendung eines Rotationskörpers ist ein Rad (hier ein Seilrad)

Partieller Rotationskörper (Grid Segment)

Makro

```
REVOLVE ELEMENT (G1, SELECTION, GRID_ROT_CONTOUR, w1, w2, nNodes)
```

g1: Grid Referenz {g1, g2, g3, ...}
SELECTION: reserviertes Wort. Es soll ein selektiertes Element als Repräsentant eines Grids benutzt werden.
GRID_ROT_CONTOUR: reserviertes Wort
w1, w2: Startwinke, Endwinkel [°]
nNodes: Anzahl Netzknoten pro 360° (die Anzahl wird autom. auf (w2-w1) Grad reduziert.

Beschreibung

Die Idee hinter dieser Funktion ist die Folgende: Eine offene oder geschlossene Linien-Bogen-Kontur wird in der X-Y-Ebene im richtigen Abstand zur X-Achse definiert (Rohdaten werden immer in der X-Y-Ebene

definiert). Das System verschiebt die Kontur bei der Ausführung automatisch in die Z-Y-Ebene und rotiert sie um die Z-Achse (sonar erzeugt Rotationskörper immer um die Z-Achse). Die Rotation muss allerdings nicht zu einer geschlossenen Ring-artigen Netzstruktur führen. Vielmehr werden mit zwei weiteren Parametern ein Start- und ein Endwinkel angegeben. Es entsteht auf diesem Weg ein Segment-artiges Gebilde, als ob man aus einem Ring-Kuchen ein Stück herausschneiden würde. Der letzte Parameter gibt die Anzahl der Zellen längs dem Umfang von 360° an. Die Funktion rechnet diesen Wert automatisch um auf das definierte Segment und erzeugt die reduzierte Anzahl von (w2-w1)/360 Gitterzellen im definierten Winkelbereich.

Die erzeugte Rohdatenstruktur ist vom Typ 'Grid' und hat folglich lediglich eine Oberfläche aber kein Volumen. Aus diesem Grund kann diese Struktur anschliessend auch nur in ein raumfestes Oberflächenobjekt ohne Volumen weiterverarbeitet werden. An den beiden Segment-Enden ist das Gebilde im übrigen stirnseitig offen. Die Funktion erzeugt so gesehen nur den Mantel der Oberfläche. Wird dieses 'Rohdaten- Grid' später in ein 'Grid-Objekt' umgewandelt, dann interagiert dieses Netz normal mit allen beweglichen Objekten welche an diese Oberfläche anstossen. Im weiteren hat das 'Grid' eine Aussen- und Innenseite welche bei der Definition der Linien-Bogen-Kontur mit sog. Normalenvektoren auf den Elementen spezifiziert wird. Die Interaktion mit anderen Objekten funktioniert nur von der Aussenseite her. Zuerst ein Beispiel:

Beispiel

```
BEGIN SCRIPT U-Profil_gebogen

-- Kontur -----------------------------------------------------
CREATE ELEMENT (E1, LINE, -1.333466, 24.65, 0, -1.333466, 21.95, 0)
SET PROPERTY (E1, NORMALVECTOR, -1, 0, 0)
CREATE ELEMENT (E2, LINE, -1.333466, 21.95, 0, -1.084034, 21.95, 0)
SET PROPERTY (E2, NORMALVECTOR, 0, -1, 0)
CREATE ELEMENT (E3, LINE, -1.084034, 21.95, 0, -1.021360, 24.15459, 0)
SET PROPERTY (E3, NORMALVECTOR, 1, 0, 0)
CREATE ELEMENT (E4, ARC, -0.8218658, 24.15046, 0, -0.8218661, 24.35, 0, -1.021360,
24.15459, 0, -1)
SET PROPERTY (E4, NORMALDIRECTION, -1)
CREATE ELEMENT (E5, LINE, -0.8218661, 24.35, 0, -0.6472, 24.35, 0)
SET PROPERTY (E5, NORMALVECTOR, 0, -1, 0)
CREATE ELEMENT (E6, LINE, -0.6472, 24.35, 0, -0.2472, 24.35, 0)
SET PROPERTY (E6, NORMALVECTOR, 0, -1, 0)
CREATE ELEMENT (E7, LINE, -0.2472, 24.35, 0, 0.2472, 24.35, 0)
SET PROPERTY (E7, NORMALVECTOR, 0, -1, 0)
CREATE ELEMENT (E8, LINE, 0.2472, 24.35, 0, 0.6472, 24.35, 0)
SET PROPERTY (E8, NORMALVECTOR, 0, -1, 0)
CREATE ELEMENT (E9, LINE, 0.6472, 24.35, 0, 0.8218661, 24.35, 0)
SET PROPERTY (E9, NORMALVECTOR, 0, -1, 0)
CREATE ELEMENT (E10, ARC, 0.8218658, 24.15046, 0, 1.021459, 24.15889, 0, 0.8218661,
24.35, 0, -1)
SET PROPERTY (E10, NORMALDIRECTION, -1)
CREATE ELEMENT (E11, LINE, 1.021459, 24.15889, 0, 1.117403, 21.95, 0)
SET PROPERTY (E11, NORMALVECTOR, -1, 0, 0)
CREATE ELEMENT (E12, LINE, 1.117403, 21.95, 0, 1.416534, 21.95, 0)
SET PROPERTY (E12, NORMALVECTOR, 0, -1, 0)
CREATE ELEMENT (E13, LINE, 1.416534, 21.95, 0, 1.416534, 24.65, 0)
SET PROPERTY (E13, NORMALVECTOR, 1, 0, 0)

-- create partial rotated grid (rawdata) ------------------------
GROUP ELEMENTS(ALL)
DESELECT ALL
SELECT ELEMENT(E1)
REVOLVE ELEMENT (G1, SELECTION, GRID_ROT_CONTOUR, 0, 25, 120)
-- das selektierte Elem. 'E1' wird um die Z-Achse gedreht
-- Es entsteht das Rohdaten-Grid mit der Referenz 'G1'
-- end of script
```

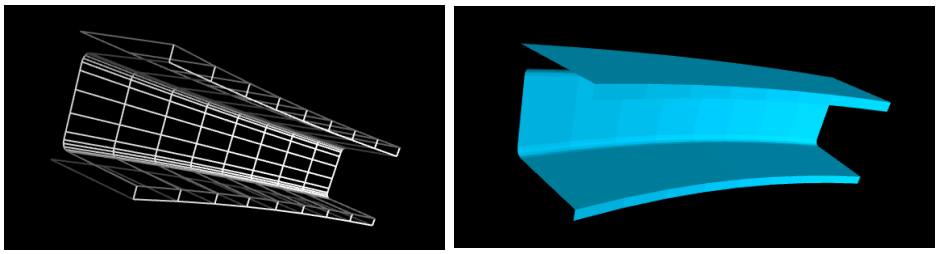
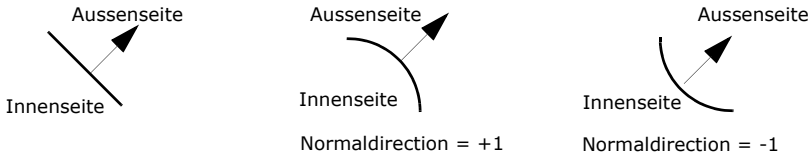


FIGURE 5. Das vom vorangehenden sonar Script erzeugte Rohdaten Grid und sein Aussehen nach der Weiterverarbeitung mit der Funktion Import Grid(). Beachten Sie, dass in der 3D-Darstellung rechts nur diejenigen Flächen in Erscheinung treten, welche von aussen her betrachtet werden.

Normalenvektoren der Elemente

Die Datenstruktur der Elemente, speziell die der Linien und Bogen, lässt es zu, dem betreffenden Element einen Normalenvektor zuzuordnen. Ein Normalenvektor sitzt per Definition mit seinem Fusspunkt senkrecht auf dem betreffenden Element und zeigt in Richtung nach Aussen.



Beachten Sie, wie im Beispiel oben angewendet, dass die Linien mit dem Parameter '`NORMALVECTOR`' gekennzeichnet werden, während die Bogen den Parameter '`NORMALDIRECTION`' benutzen. Diese Unterscheidung kommt von daher, dass beim Bogen die Richtung des Normalenvektors längs dem Bogen ständig ändert und man deshalb nicht von einem bestimmten Normalenvektor sprechen kann. Aus diesem Grund spricht man beim Bogen von einer positiven oder negativen Richtung in Bezug zum Radiusvektor des Bogens.

Weiterverarbeitung des Rohdaten-Grids

Ein Rohdaten-Grid nimmt wie alle anderen Rohdaten an den Simulationen nicht teil. Damit ein Grid in die Lage versetzt wird, physikalisch zu wirken, muss es in ein Objekt weiterverarbeitet werden. Sehen Sie dazu den nächsten Abschnitt 'Grid Surface'.

Grid Surface

Makro

```
IMPORT GRID (G1, FILENAME, "filename", typeNr)
```

G1: Objekt Referenz {o1, o2, o3, ...} des erzeugten Grid-Objektes
FILENAME: reserviertes Wort
"filename": Ein regulärer Filename zwischen Anführungs- und Schlusszeichen.
typeNr: Bezeichner für einen bestimmten grid-Typ

Beschreibung

Eine Netzstruktur kann nicht direkt mit einem Befehl erzeugt werden, sondern ist auf den Import einer Beschreibung der einzelnen Knoten des Netzes angewiesen. Betreffend dieser Beschreibung gibt es mehrere Methoden, diese Knotenstruktur und ihre Koordinaten festzulegen. Sehen Sie dazu auch das user-manual.
Die Funktion importiert eine Rohdaten-Netzstruktur und wandelt diese in ein physikalisches Objekt vom Typ 'Grid' um. Das Grid-Objekt ist schliesslich eine raumfeste, unbewegliche Berandung welche mit allen Objekten normal interagiert.

Grid Nachbearbeitung (Hilfsfunktionen)

In sonar-LAB gibt es einige Funktionen um Primitivkörper vom Typ 'Grid', wie sie vorerst mit den Standardfunktionen erzeugt wurden, nachträglich zu ändern. Es handelt sich meistens um sehr spezielle Funktionen für die Durchführung bestimmter Aufgaben. Allerdings können mit diesen Funktionen nur Rohdaten-Grids verändert und modifiziert werden.

`DEFORM GRID (G1, ALIGNED, X || Y || Z, min, max)` **Makro**

g1: Grid Referenz {g1, g2, g3, ...}eines Rohdaten-Grids
ALIGNED: reserviertes Wort -> bedeutet in diesem Zusammenhang = fluchtend.
Y: Deformationsrichtung
min, max: Begrenzungen der Scheibe, aus der das Netz entfernt bzw. zurückgedrängt werden soll.

Beschreibung

eine vorher erzeugte Grid-Struktur mittels der Funktion 'REVOLVE SECTION wird in eine bestimmte Hauptrichtung eingedrückt bzw. zurückgedrängt. Die dabei betroffenen Grid-Bereiche werden dabei in Richtung Netzlinien zurückversetzt.

`SET POINT (G1, hIdx, vIdx, X | Y | Z,` *double*`)` **Makro**

g1: Grid Referenz {g1, g2, g3, ...}
hIdx, vIdx: Knoten-Indizes des Netzes
X | Y | Z: Verschiebungsrichtung
double: Koordinatenwert

`SET POINT (G1, hIdx, vIdx, x1, y1, z1)` **Beschreibung**

g1: Grid Referenz {g1, g2, g3, ...}
hIdx, vIdx: Knoten-Indizes des Netzes
x1, y1, z1: Koordinatenwerte

Die Funktion 'SET POINT' erlaubt das nachträgliche Verschieben einzelner Knotenpunkte in eine bestimmte Koordinatenrichtung. Während die erste der beiden Funktionen den spezifizierten Punkt nur in eine bestimmte Koordinatenrichtung verschiebt, setzt die zweite Funktion die Position auf einen beliebigen räumlichen Punkt.

Sweep

`SWEEP CROSSSECTION(O1,SELECTION, POLYLINE || LINE_ARC, CIRCLE, R)` **Makro**

o1: Objekt Referenz {o1, o2, o3, ...}
SELECTION: reserviertes Wort. Es soll ein selektiertes Element benutzt werden.
POLYLINE || LINE_ARC: das selektierte Element ist eine Polyline oder eine Kombination von Linien und Bogen.
CIRCLE: reserviertes Wort: der Querschnitt des ausgezogenen Profils ist ein Kreis.
R: der konstante Radius des Querschnittes

Beschreibung:

Mit der Sweep Funktion lassen sich beliebig verbogene zylindrische Stangen erzeugen. Ein zylindrisches Gebilde in Form eines Zapfenziehers wäre ein Beispiel für ein solches Objekt. Ein Kneter in einer Knetmaschine wäre ein weiteres Beispiel.

Eigenschaften (Primitives)

Die Eigenschaften der ‘Primitives’ lassen sich alle einzeln setzen und aktivieren. Alle Eigenschaften sind vom Typ

```
SET PROPERTY (objectReference, parameterList)
```

Es gilt zu beachten, dass in einem script nur die Eigenschaften gesetzt werden müssen, die erstens relevant sind und benutzt werden und zweitens von den default-Eigenschaften abweichen. So ist z.B. die Anfangs-geschwindigkeit eines Objektes per default für alle Koordinatenrichtungen gleich Null. In der Regel ist das in Ordnung so, und muss nicht nochmals explizit gesetzt werden.
Auch in allen folgenden Anweisungen gilt es zu beachten, dass das [cm-g-µs]-Einheitensystem strikte eingehalten wird.

Ueberblick

Physik

```
SET PROPERTY (O1, ANGULAR_VELOCITY, {X|Y|Z}, double)
SET PROPERTY (O1, ANGULAR_VELOCITY, vx, vy, vz)
SET PROPERTY (O1, DENSITY, double)
SET PROPERTY (O1, FORCE_EXT, {X|Y|Z}, double)
SET PROPERTY (O1, FRICTION_UNILATERAL, double)
SET PROPERTY (O1, MASS, double)
SET PROPERTY (O1, MOMENT_FORCE_EXT, nx, ny, nz, double)
SET PROPERTY (O1, MOMENT_INERTIA, Ix, Iy, Iz)
SET PROPERTY (O1, MOMENT_INERTIA, FACTOR, double)
SET PROPERTY (O1, ROTATION_LOCKED, X|Y|Z, bool)
SET PROPERTY (O1, SIM_MEMBER, bool)
SET PROPERTY (O1, VELOCITY, {X|Y|Z}, double)
```

Interaktion

```
SET PROPERTY (O1, C_INTERACT_LIN, double)
SET PROPERTY (O1, C_INTERACT_QUAD, double)
SET PROPERTY (O1, INTERACT_CONTROLPOINT, x1, y1, z1, bool)
SET PROPERTY (O1, INTERACT_DIRECTION, directionSpecifier)
SET PROPERTY (O1, INTERACT_METHOD, ELASTIC, double)
SET PROPERTY (O1, INTERACT_MODE, {ACTIVE,PASSIVE,NO_INTERACTION})
```

Allgemein

```
SET PROPERTY (O1, BEVEL, ROUND || FACET, double)
SET PROPERTY (O1, COLOR_STD, integer)
SET PROPERTY (O1, COLOR_RGB, red, green, blue)
SET PROPERTY (O1, GROUP_NR, LAST_GROUP_NR)
SET PROPERTY (O1, NAME, “objectname”)
SET PROPERTY (O1, SUPERGROUP_NR, LAST_SUPERGROUP_NR)
SET PROPERTY (O1, TRANSPARENCY, integer)
SET PROPERTY (O1, VISIBILITY, bool)
SET PROPERTY (O1, WIREFRAME, bool)
```

Winkelgeschwindigkeit (angular velocity)

Syntax 1

```
SET PROPERTY (O1, ANGULAR_VELOCITY, {X|Y|Z}, double)
```

o1: Objekt Referenz {o1, o2, o3, ...}
ANGULAR_VELOCITY: reserviertes Wort
{X|Y|Z}: Es wird eine Koordinatenrichtung gesetzt (X oder Y oder Z)
double: Fliesskommawert

Syntax 2

```
SET PROPERTY (O1, ANGULAR_VELOCITY, vx, vy, vz)
```

o1: Objekt Referenz {o1, o2, o3, ...}
ANGULAR_VELOCITY: reserviertes Wort
vx, vy,vz: es werden alle 3 Koordinaten des Winkelgeschwindigkeitsvektors gesetzt
double: Fliesskommawert

Beschreibung

Die Winkelgeschwindigkeit ist ein Vektor, welcher in der aktuellen Drehachse des betreffenden Objektes liegt und dessen Länge den Betrag der Winkelgeschwindigkeit angibt. Man kann sich vorstellen, dass dieser Vektor im Schwerpunkt des Objektes angeheftet ist. Mit dieser Anweisung bekommt entweder eine Komponente {X|Y|Z} oder der gesamte Vektor der Winkelgeschwindigkeit des Referenzobjektes einen neuen Wert. Wird lediglich eine Komponente des Winkelgeschwindigkeitsvektors neu gesetzt, dann behalten die beiden anderen Komponenten den Wert den sie im Moment haben.

Beispiel

```
SET PROPERTY(O7, ANGULAR_VELOCITY, -4.0E-6, 3.0E-6, 0)
```

Erzeugt wurde damit ein Winkelgeschwindigkeitsvektor in der X-Y-Ebene mit einer Länge von 5.0E-6 rad/µs = 5.0 rad/s = 0.8 Umdrehungen/s. Sofern der dargestellte Zylinder keine weiteren Verbindungen hätte und frei im Raum schweben würde, hätte diese Initialisierung eine Taumelbewegung des Zylinders zur Folge.

Dichte (density)

Syntax

```
SET PROPERTY (O1, DENSITY, double)
```

o1: Objekt Referenz {o1, o2, o3, ...}
DENSITY: reserviertes Wort
double: Fliesskommawert > 0

Beschreibung

Die Funktion setzt die Dichte des referenzierten Objektes auf einen neuen Wert. Weitere Variablen die von der Dichte abhängig sind, werden automatisch neu berechnet (Masse, Trägheitsmoment).

Beispiel

```
SET PROPERTY (O34, DENSITY, 7.8)
```

Das Objekt welches in einem script als ‘O34’ benannt wurde, bekommt die Dichte von Stahl, also 7.8 g/cm^3.

Externe Kraft (external force)

Syntax

```
SET PROPERTY (O1, FORCE_EXT, {X|Y|Z}, double)
```

o1: Objekt Referenz {o1, o2, o3, ...}
FORCE_EXT: reserviertes Wort

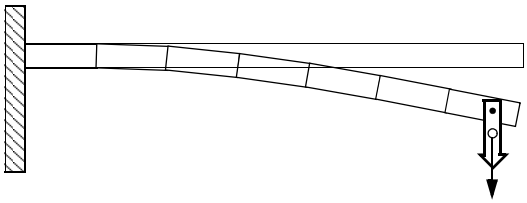
{X|Y|Z}: Es wird eine Koordinatenrichtung gesetzt (X oder Y oder Z)
double: Fliesskommawert

Beschreibung

Die Anweisung setzt einen Kraftbetrag für die sog. externe Kraft des betreffenden Objektes in einer bestimmten Koordinatenrichtung. Durch zwei- oder dreifache Anwendung dieser Anweisung für die anderen beiden Koordinatenrichtungen kann auf diesem Weg auch eine beliebige vektorielle Kraft definiert werden. Jedes Objekt bietet in seinem Speicher Platz für eine beliebige, nicht näher spezifizierte externe Kraft, welche konstant und vektoriell auf den Schwerpunkt des Objektes wirkt. Unabhängig davon, welche räumliche Lage das Objekt im Moment auch haben mag, die Wirkungsrichtung ist immer dieselbe und bezieht sich auf das feste globale Koordinatensystem. Die hier gesetzte externe Kraft ist identisch mit der Kraft gleichen Namens wie sie im Dialog 'Object Properties' manuell gesetzt werden kann. Hier wie dort bleibt diese Kraft aktiv, bis sie durch eine andere oder Null ersetzt wird. Die externe Kraft wird letztlich auch zusammen mit dem Modell gespeichert.

Beispiel

Eine typische und allgemeine Anwendung für eine konstante externe Kraft zu finden ist nicht ganz einfach und eher problemabhängig. Deshalb zuerst, was eine externe Kraft nicht ist. Sie ist kein Ersatz für eine Kraft, denn diese ist von der Masse des Objektes abhängig (F = m*g). Man könnte aber eine Störung als externe Kraft definieren, indem eine kleine externe Kraft wie eine 'Drift' immer in eine bestimmte Richtung zieht.
Eine andere Anwendung ist eine externe Belastung mit einer bestimmten Kraft, welche man auf eine elastische Struktur ausüben möchte. So könnte man eine biegsamen elastischen Balken an einem Ende fest einspannen und am anderen Ende mit einer konstanten Kraft belasten um dessen Verbiegung zu messen.



In diesem Beispiel wäre der weisse Pfeil am Ende des Balkens ein wirkliches physikalisches Objekt, welchem man zur Klarstellung die Form eines Pfeils gegeben hätte. Der Pfeil wäre mit einem Link am letzten Element des Balkens befestigt und besitzt in seinem Schwerpunkt eine externe Kraft, welche ihn vertikal nach unten zieht.

Unilaterale Reibung (unilateral friction)

Syntax

```
SET PROPERTY (O1, FRICTION_UNILATERAL, μ)
```

o1: Objekt Referenz {o1, o2, o3, ...}
FRICTION_UNILATERAL: reserviertes Wort
μ: linearer Reibungskoeffizient (Fliesskommawert >= 0)

Beschreibung

Während sich eine bilaterale Reibung nur auf zwei einzelne, ausgewählte Objekte bezieht, betrifft eine unilaterale Reibung ein einzelnes Objekt in Verbindung mit allen anderen Objekten welche dieses Objekt berühren. Die unilaterale Reibung kommt nach Prioritäten geordnet nach der bilateralen Reibung an zweiter Stelle und vor der allgemeinen globalen Reibung.

- 1. bilaterale Reibung

- 2. unilaterale Reibung
- 3. globale Reibung

Das heisst in Worten, dass das Programm bei jeder Objekt-Begegnung zuerst nachschaut ob eine bilaterale Regel für diese Paarung vorliegt. Wenn ja, dann wird diese angewendet. Andernfalls wird in einem zweiten Schritt geprüft, ob eines der beiden Objekte eine unilaterale Regel hat. Haben beide Objekte eine unilaterale Regel, dann wird der Mittelwert genommen. Hat keines der Objekte eine Regel, dann wird schliesslich die globale Reibung zur Anwendung gebracht. All diese Reibungen kommen im Uebrigen nur dann zur Wirkung, wenn die Reibung aktiviert wurde.

Beispiel

```
SET PROPERTY (O16, FRICTION_UNILATERAL, 0.14)
```

Für alle Objektpaarungen wo das Objekt mit der Referenznummer 'O16' beteiligt ist, wird ein Reibungskoeffizient von 0.14 angewendet.

Masse (mass)

Syntax

```
SET PROPERTY (O1, MASS, double)
```

o1: Objekt Referenz {o1, o2, o3, ...}
MASS: reserviertes Wort
double: Fliesskommawert > 0

Beschreibung

Die Masse eines Objektes in Gramm. Alle Variablen welche von der Masse abhängen werden in der Folge automatisch neu berechnet. Dazu gehören die Dichte und das Trägheitsmoment.

Ext. Drehmoment (ext. moment of force)

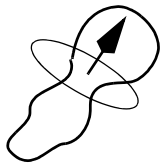
Syntax

```
SET PROPERTY (O1, MOMENT_FORCE_EXT, nx, ny, nz, double)
```

o1: Objekt Referenz {o1, o2, o3, ...}
MOMENT_FORCE_EXT: reserviertes Wort
nx, ny, nz: die drei Komponenten eines Momentenvektors
double: Fliesskommawert > 0

Beschreibung

Die drei Variablen (nx, ny, nz) bestimmen den Richtungsvektor beliebiger Länge des Drehmomentes. Diese drei Vektor-Komponenten werden anschliessend vom Programm automatisch auf einen Einheitsvektor zurückskaliert welcher die Länge Eins besitzt. Dieser Vektor ist ein Normalenvektor welcher im Objektschwerpunkt angeheftet ist und senkrecht auf der Drehebene steht. Der letzte Parameter bestimmt letztlich den eigentlichen Drehmomentbetrag während der Vektor (nx, ny, nz) nur die Richtung vorgibt. Das definierte Drehmoment bleibt so lange wirksam, bis es durch einen anderen Wert oder Null ersetzt wird. Das Drehmoment wird mit dem Modell abgespeichert.



Beispiel

```
SET PROPERTY (O4, MOMENT_FORCE_EXT, 0, 0, 1, 1.25E-5)
```

Der Drehmomentvektor zeigt in Richtung der Z-Achse und hat den Wert von 1.25E-5 [g cm2/µs2], was umgerechnet 1.25 Nm im metrischen System entspricht.

Trägheitsmoment (moment of inertia)

Syntax

```
SET PROPERTY (O1, MOMENT_INERTIA, Ix, Iy, Iz)
```

o1: Objekt Referenz {o1, o2, o3, ...}
MOMENT_INERTIA: reserviertes Wort
Ix, Iy, Iz: die drei Komponenten des Trägheitsmoments des Objektes

Beschreibung

Die Anweisung berechnet und setzt die drei Massen- und Form-abhängigen Komponenten des Massenträgheitsmoments für das Objekt welches mit der Referenz 'O1' erzeugt wurde.

Beispiel

Für einen Quader mit der Masse m und den Kantenlängen a, b und c würden für das Trägheitsmoment die folgenden Werte gesetzt:

$Ix = m(b^2+c^2)/12, \quad Iy = m(a^2+c^2)/12, \quad Iz = m(a^2+b^2)/12$

Syntax

```
SET PROPERTY (O1, MOMENT_INERTIA, FACTOR, double)
```

o1: Objekt Referenz {o1, o2, o3, ...}
MOMENT_INERTIA: reserviertes Wort
FACTOR: reserviertes Wort
double: ein Multiplikationsfaktor > 0 für das Trägheitsmoment des Objektes

Beschreibung

Ein bereits gesetztes Trägheitsmoment kann zu einem späteren Zeitpunkt modifiziert werden, indem es mit einem positiven Faktor multipliziert wird. Diese Funktion macht Sinn für Untersuchungen mit unterschiedlichen Trägheiten ohne sich vorerst auf bestimmte Materialien festlegen zu wollen. Auch in Zusammenhang mit Toleranzauswirkungen kann diese Funktion zur Anwendung kommen.

Rotationsachsen einfrieren

Syntax

```
SET PROPERTY (O1, ROTATION_LOCKED, {X|Y|Z}, bool)
```

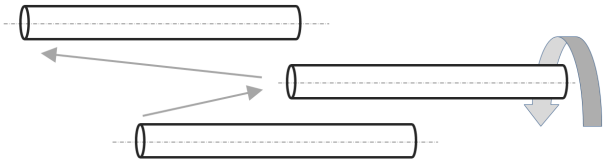
o1: Objekt Referenz {o1, o2, o3, ...}
ROTATION_LOCKED: reserviertes Wort
{X|Y|Z}: Es wird eine Koordinatenrichtung gesetzt (X oder Y oder Z)
bool: TRUE oder FALSE (für aktivieren oder deaktivieren)

Beschreibung

Mit dieser Funktion wird eine einzelne lokale Rotationsachse eines bestimmten Objektes 'eingefroren'. Dies bedeutet, dass das Objekt um die betreffende Achse keinerlei Rotationen mehr durchführt. Die anderen Achsen des Objektes bleiben aber aktiv, sofern für diese nicht die gleichen Einschränkungen gesetzt wurden. Das Wort 'lokal' ist in diesem Zusammenhang wichtig.

Beispiel

Nehmen wir an, in einem Zylinder, wie abgebildet, werden die X- und die Y-Achse eingefroren, In der Folge kann sich dieser nur noch um die Z-Achse drehen. Und die Z-Achse ist bei einem Zylinder immer die Zylinderachse, unabhängig davon, welche Stellung der Zylinder im Moment im Raum hat.



Dies bedeutet letztlich, dass die translatorischen Bewegungen im Raum weiterhin normal durchgeführt werden, das Objekt wird sich um die X- und Y-Achse einfach nicht mehr drehen, aber um seine eigene Achse wird das Objekt sich weiterhin frei drehen können.

Räumlich fixierte Objekte

Syntax

```
SET PROPERTY (O1, SIM_MEMBER, bool)
```

o1: Objekt Referenz {o1, o2, o3, ...}
SIM:MEMBER: reserviertes Wort
bool: TRUE oder FALSE (für aktivieren oder deaktivieren)

Beschreibung

Das verwendete Synonym für den Begriff, dass ein Objekt im Raum fixiert bleibt und sich folglich überhaupt nicht mehr dreht und bewegt heisst 'SIM_MEMBER' (Simulations Teilnehmer). Ein Objekt welches an der Simulation nicht mehr teilnimmt wird bei den Berechnungen sämtlicher Bewegungen übergangen. Das Objekt ist aber weiterhin vorhanden, existent und nimmt sogar an den Kollisionen weiterhin teil, so wie diese Eigenschaften für das betreffende Objekt ggf. definiert sind. So gesehen ist die Aussage, dass das Objekt an den Simulationen nicht mehr teilnimmt, vielleicht etwas zu krass ausgedrückt, aber wir bleiben jetzt bei diesem Ausdruck.

Beispiel

```
SET PROPERTY (O30, SIM_MEMBER, FALSE)
```

Das Objekt mit der Referenz-Nr. 30 ist im Raum fixiert.

Geschwindigkeit (velocity)

Syntax

```
SET PROPERTY (O1, VELOCITY, {X|Y|Z}, double)
```

o1: Objekt Referenz {o1, o2, o3, ...}
VELOCITY: reserviertes Wort
{X|Y|Z}: Es wird eine Koordinatenrichtung gesetzt (X oder Y oder Z)
double: Fliesskommawert

Beschreibung

Die translatorische Geschwindigkeit des Schwerpunktes eines Objektes wird mit dieser Anweisung für jede Raumkoordinate einzeln zugewiesen. Natürlich bezieht sich diese Geschwindigkeit auf das globale Koordinatensystem. Diese Funktion wird häufig dazu verwendet um einem Objekt eine Anfangs-geschwindigkeit zu geben oder um die Geschwindigkeit eines Objektes in einem 'sonar script control system' kontinuierlich zu steuern.

Zylinder-Facette (bevel)

Syntax

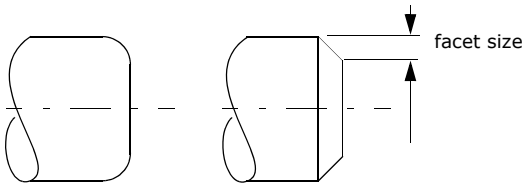
```
SET PROPERTY (O1, BEVEL, ROUND || FACET, double)
```

o1: Objekt Referenz {o1, o2, o3, ...}
BEVEL, ROUND, FACET: reservierte Worte
double: facet size

Beschreibung

Diese Funktion ist zum Nachbearbeiten eines bereits bestehenden Zylinders mit der Referenz 'O1'. Der ursprünglich scharfkantige Zylinder bekommt an seinen kreisförmigen Kanten entweder eine Facette von 45° oder eine

Rundung je nachdem wie der dritte Parameter gesetzt wird. Die Abmessung der Facette bzw. Rundung wird mit dem letzten Parameter übergeben.



Die Facetten nehmen als geometrische Konturen an den Interaktions- und Kollisionsberechnungen während den Simulationen teil.

Objekt Farbe

Syntax

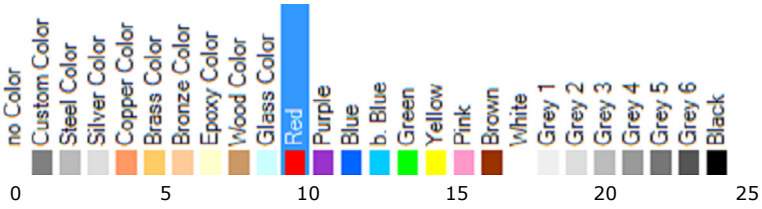
```
SET PROPERTY (O1, COLOR_STD, nr)
SET PROPERTY (O1, COLOR_RGB, red, green, blue)
```

o1: Objekt Referenz {o1, o2, o3, ...}
COLOR_STD, COLOR_RGB: reservierte Worte
nr: integer [0..25]
red, gree, blue: integer Werte [0..255]

Beschreibung

Einem Objekt kann eine beliebige Farbe zugeordnet werden. Es gibt dazu zwei Möglichkeiten. Die erste der beiden Varianten greift per Nummer auf eine vordefinierte Standardfarbe zurück, die zweite Variante definiert die Farbe explizit mit den drei Komponenten (rot, grün, blau). Jede Komponente ist eine Zahl im Bereich [0..255] wie sie auch das Farb-Tool des Systemprogramms 'Paint' benutzt (siehe: Paint / Palette bearbeiten).

Die Standardfarben, wie untenstehend abgebildet sind von links nach rechts durchnummeriert. Beginnend mit '0' für 'no color' besitzt die Farbe 'Red' die Nummer '10' und die letzte Farbe 'Black' schliesslich '25'.
Diese Standardfarben stehen auch im Dialog 'Edit Objekt Property' zur Verfügung.



Gruppenzugehörigkeit

Syntax

```
SET PROPERTY (O1, GROUP_NR, LAST_GROUP_NR)
SET PROPERTY (O1, SUPERGROUP_NR, LAST_SUPERGROUP_NR)
```

o1: Objekt Referenz {o1, o2, o3, ...}
GROUP_NR, SUPERGROUP_NR: welcher Art von Gruppe soll das Objekt hinzugefügt werden
LAST_GROUP_NR, LAST_SUPERGROUP_NR: indirekte Referenzierung der Gruppennummer.

Beschreibung

Objekte können zu Objektgruppen hinzugefügt werden. Es gibt vorerst zwei Hierarchiestufen von Gruppen: (normale) Gruppen und Supergruppen. Letztere sind Gruppen von Gruppen. Die Gruppen- bzw. Supergruppennummer muss bereits bestehen. Mit diesen beiden Anweisungen wird ein Objekt immer an die letzte Gruppe bzw. Supergruppe angefügt. Es gibt dazu auch eine Funktion um jederzeit in einem script eine neue Gruppennummer zu erzeugen. Sehen Sie dazu die Anweisungen

```
SET VALUE (NEW_GROUP_NR || NEW_SUPERGROUP_NR)
```

Man benutzt in diesem Zusammenhang also nicht eine absolute sondern eine relative Referenzierung zur Gruppennummer, indem man ggf. im script einfach eine neue Nummer erzeugt und alle relevanten Objekte an diese anfügt. Die absolute Referenzierung ist wenig zuverlässig und kann sich während der Bearbeitung ändern. Dies trifft z.B. zu, wenn mehrere Modelldateien 'ge-merged' bzw. zusammengefügt werden, welche vorher teilweise gleiche Gruppennummern verwendeten. Um solchen Konflikten aus dem Weg zu gehen bekommen Gruppen beim 'mergen' oft automatisch neue Gruppennummern.

Objektname

Syntax

```
SET PROPERTY (O1, NAME, "objectname")
```

o1: Objekt Referenz {o1, o2, o3, ...}
NAME: reserviertes Wort [max. 32 Zeichen]
"objectname": In Gänsefüsschen gesetzter Name des Objektes

Beschreibung

Ein beliebiger, vom Benutzer gesetzter, Name des Objektes mit maximal 32 Zeichen Länge. Ueberzählige Zeichen werden einfach abgeschnitten. Der Name muss kein zusammenhängendes Wort sein. Diese Bezeichnung wird in den sog. 'Interaction Rules by Name'- Regeln zum Vergleich herangezogen. Im 'Object Tool' werden die Objekte unter diesen Namen aufgeführt. Letztlich wird dieser Name in vielen Funktionen und Dialogen zur schnelleren Identifikation gezeigt. Die Namensgebung ist aber nicht obligatorisch. Wird kein Name vergeben, dann wird oft die Bezeichnung 'not specified' verwendet bzw. angezeigt.

Sichtbarkeit

Syntax

```
SET PROPERTY (O1, VISIBILITY, bool)
```

o1: Objekt Referenz {o1, o2, o3, ...}
VISIBILITY: reserviertes Wort
bool: Aktivierung der Sichtbarkeit (TRUE, FALSE)

Beschreibung

Objekte können in den Darstellungen am Bildschirm ausgeblendet werden. Objekte welche die Eigenschaft (visibilty = FALSE) haben, werden beim Zeichnen in den 2D- und 3D-Ansichten einfach übergangen. Darüber hinaus bleiben diese ausgeblendeten Objekte aber weiterhin in jeder Hinsicht aktiv. Insbesondere nehmen sie an den Simulationen ohne Einschränkungen teil.

Wireframe

Syntax

```
SET PROPERTY (O1, WIREFRAME, bool)
```

o1: Objekt Referenz {o1, o2, o3, ...}
WIREFRAME: reserviertes Wort

bool: Aktivierung der Wireframe Darstellung des Objektes (TRUE, FALSE)

Beschreibung

Ein Objekt kann in der 3D-Darstellung statt als ‘solid’ auch als ‘wireframe’-Objekt dargestellt werden (Drahtgitterdarstellung). Ein Objekt welches auf diese Weise gezeichnet wird, ist durchsichtig. Dies ist mitunter ein Grund, weshalb man einzelne Objekte manchmal als wireframe darstellt. Es werden nur die Kanten gezeichnet, aber ohne Flächen. Schaltet man die Sichtbarkeit (Visibility) eines Objektes aus, dann ist von ihm nichts mehr zu sehen. Als Wireframe kann man das Objekt und seine Lage aber weiterhin erkennen.

Objekt-Interaktion

Interaktionsregel Erzeugen

Makro 1

```
CREATE IACT_RULE (O1, O2, bool)
```

O1, O2: Objekt Referenz {o1, o2, o3, ...}
bool: bool'scher Wert {TRUE || FALSE}

Makro 2

```
CREATE IACT_RULE (O1, O2, bool, mode1, mode2)
CREATE IACT_RULE (SELECTION, bool, mode1, mode2)
```

O1, O2: Objekt Referenz {o1, o2, o3, ...}
bool: bool'scher Wert {TRUE || FALSE}
mode1, mode2: {SINGLE || GROUP_NR || SUPERGROUP_NR}

Beschreibung

Der bool'sche Wert setzt die eigentliche Interaktionsregel fest, welche besagt: Ja, die beiden Objekte sollen interagieren oder Nein, sie sollen es nicht.
Die Parameter '`mode1`' und '`mode2`' machen eine Aussage über den Anwendungsbereich dieser Regel: Bezieht sich die Regel bezüglich '`O1`' resp. '`O2`' jeweils über ein einzelnes Objekt, über die ganze Gruppe welcher '`O1`' angehört oder sogar die ganze Supergruppe. Dementsprechend kann der Benutzer mit diesen zusätzlichen Parametern für beide Referenz-Objekte eine entsprechende Information einbringen. Wie diese Regel auch immer gesetzt wird, es handelt sich in jedem Fall um eine bilaterale Interaktionsregel. Wird die Anweisung auf eine ganze Gruppe angewendet, dann werden entsprechend mehrere bilaterale Regeln aufgesetzt, für jede Paarung eine.

Interaktionskonstante (interaction const.)

Syntax

```
SET PROPERTY (O1, C_INTERACT_LIN, double)
SET PROPERTY (O1, C_INTERACT_QUAD, double)
```

o1: Objekt Referenz {o1, o2, o3, ...}
C_INTERACT_LIN, C_INTERACT_QUAD: reserviertes Wort
double: Fliesskommawert

Beschreibung

Die Interaktionskonstante bestimmt die Oberflächenhärte bei Kollisionen. Man kann sich diese als Federkonstante vorstellen. Objekte können weich wie Gummi oder hart wie Stahl kollidieren. Ein guter Mittelwert für Problemstellungen im Labormassstab ist '0.01'. Kleinere Werte, z.B. '0.0001', charakterisieren weichere Materialien, grössere Werte z.B. '1.0' entsprechend härtere. Die Interaktionsfunktion rechnet immer mit zwei Konstanten, einer linearen und einer quadratischen. Wie bei einer Druckfeder entstehen bei der Kollision von zwei Objekten abstossende Kräfte nach der Formel

$$F_r = \pm f_H * (c_{LIN} * dr + c_{QUAD} * dr^2 + D_r)$$

F_r: abstossende Kraft

c_{LIN}, c_{QUAD}: Interaktionskonstanten
dr: Kollisionstiefe zwischen den Objekten
f_H: Hysterese Faktor (berücksichtigt die Energie Absorption beim Stoss)
D_r: Dämpfungsterm (numerische Dämpfung)

Die berechnete Kraft wirkt immer und ausnahmslos auf beide Objekte mit dem gleichen Betrag in entgegengesetzter Richtung (actio = reactio). Damit wird letztlich die Impulserhaltung gewährleistet.

Die beiden Interaktionskonstanten (linear, quadratisch) können unabhängig voneinander mit einer der beiden Anweisungen gesetzt werden. Der Benutzer hat damit die Möglichkeit, eher ausgeglichene Wertpaarungen zu setzen oder das Gewicht mehr auf eine der beiden Konstanten zu legen um die Charakteristik der Stossberechnung entsprechend zu beeinflussen.

Interaktionspunkte

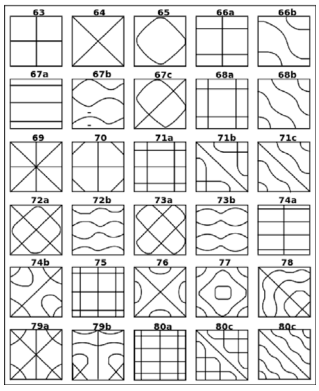
Syntax

```
SET PROPERTY (O1, INTERACT_CONTROLPOINT, x1, y1, z1, bool)
```

o1: Objekt Referenz {o1, o2, o3, ...}
INTERACT_CONTROLPOINT: reserviertes Wort
{x1|y1|z1}: die kartesischen Koordinaten des Kontrollpunktes im lokalen, Objekt-eigenen Koordinatensystem
bool: Aktivierung des Kontrollpunktes (TRUE, FALSE)

Beschreibung

Grundsätzlich kümmert sich das sonar System selbständig um die Berechnungen der Interaktionen zwischen den Objekten. Der Benutzer spezifiziert welche Objekte wann und wie miteinander kollidieren sollen, aber die Art und Weise wie die Interaktion eines gewissen Objektes mit einem anderen Objekt im Einzelnen berechnet werden soll, das erledigt die Software selbständig. Es gibt aber Fälle, wo der Benutzer dem System helfen kann, diese Berechnung besser, gezielter oder schneller durchzuführen. Ein solches Mittel in diese Berechnungen einzugreifen ist das Setzen von sog. Kollisionspunkten, die letztlich nichts anderes bewirken, als dem System mitzuteilen, wo genau die Interkation zwischen zwei Objekten am besten berechnet werden soll. Diese Information einzubringen macht nur in speziellen Begegnungsarten zwischen den Objekten einen Sinn bzw. ist nur in speziellen gegenseitigen Anordnungen der Objekte zweckmässig.



Chladnische Klangfiguren entstehen auf schwingenden Blechen mit aufgestreutem Sand

Ein Beispiel für den Einsatz dieser Eigenschaft ist dann gegeben, wenn zwei oder mehrere flache, scheibenartige oder blechartige Objekte mit ihrer gesamten Oberfläche aufeinander liegen. Man kann leicht verstehen, dass die Interaktionskontrolle in einer solchen Situation deshalb nicht trivial ist, weil der Eindringvorgang einer Scheibe in die andere infinitesimal gesehen in einem Moment die gesamte Fläche umfassen kann, wenige Rechenzyklen später aber nur noch einen kleinen Eckbereich betrifft, weil der Rest sich für einen kleinen Moment um ein paar Nanometer vom anderen Blech entfernt hat. Solchermassen grossflächig interagierende Objekte tendieren dazu, ihre Art der Interaktion dynamisch ständig zu ändern und neigen deshalb zu Schwingungen. Sie bringen Unruhe in das System. Die Ursache für das Uebel ist die ständig ändernde Interaktionsfläche welche stetig zwischen den Objekten herumwandert wie chladnische Klangfiguren. Mit dem Setzen von definierten Interaktionspunkten wirkt man genau dieser Ursache entgegen, indem man festlegt, dass die Interaktion zwischen zwei Blechteilen immer an gleichen Orten stattfinden soll. Dies führt letztlich zu wesentlich stabileren Interaktionen. Das Festlegen der Positionen von Kontrollpunkten liegt in der Verantwortung des Benutzers und muss wohl überlegt sein. Die Punkte sollten unter Berücksichtigung der zu erwartenden Bewegungen der Objekte im Laufe einer Simulation ihren Kontakt zum Zielobjekt mit Vorteil nicht verlieren, d.h. nicht über die Kanten hinaus geraten.

Ein Beispiel, wo der Einsatz dieser Funktion angebracht ist, finden wir bei einem sog. Zahnkettentrieb, wo mehrere Kettenglieder nebeneinander liegen und die Zwischenglieder sich zudem seitlich frei bewegen können und nur von den Nachbargliedern in ihrer seitlichen Bewegungsfreiheit eingeschränkt werden.

Beispiel

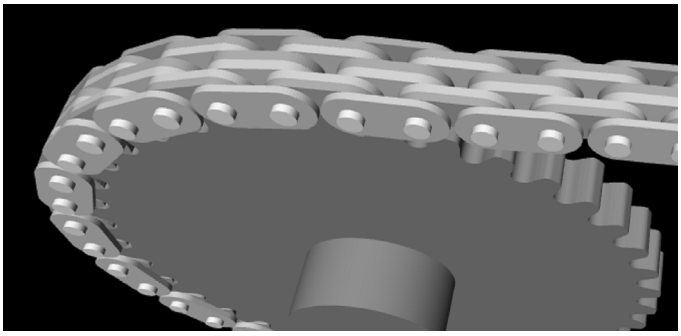


FIGURE 6. Zahnkettentrieb mit gestapelten Laschen

FIGURE 7. Die Interaktionspunkte über und unter den Bohrungen einzelner Laschen welche den Ort der seitlichen Interaktion zu den Nachbarlaschen vorgeben.

Interaktionsrichtung einschränken

Syntax

```
SET PROPERTY (O1, INTERACT_DIRECTION, directionSpecifier)
```

o1: Objekt Referenz {o1, o2, o3, ...}
INTERACT_DIRECTION: reserviertes Wort
directionSpecifier: {ALL_DIRECTIONS, AXIAL_ONLY, RADIAL_ONLY}
default: ALL_DIRECTIONS

Beschreibung

Die Interaktion einzelner Objekte kann bezüglich der Raumrichtung, in der sie berechnet werden soll, eingeschränkt werden. Es geht dabei um eine Leistungssteigerung. Der Begriff 'Raumrichtung' bezieht sich dabei auf die Begriffe 'axial' und 'radial', welche in Bezug zur Nullstellung der Objekte zu interpretieren sind. Wenn wir davon ausgehen, dass die beiden Laschen in der letzten Abbildung in ihrer Nullstellung dargestellt sind, dann liegen diese in der X-Y-Ebene und die Achse senkrecht zur Bildschirmebene ist die Z-Achse. Eine Bewegung einer Lasche in Z-Richtung heisst in dieser Terminologie immer 'axial' und eine Bewegung in der X-Y-Ebene 'radial'. Wenn es um diese Begriffe geht, können wir uns folglich immer eine Achse in Z-Richtung vorstellen auf der z.B. ein Rad aufgeschoben wurde, welches 'axial' verschoben oder 'radial' belastet werden kann.
In Zusammenhang mit diesen Begriffen ist auch ersichtlich, dass diese Funktion nur für Objekte einen Sinn macht, welche ihre räumliche Ausrichtung in Bezug zu diesen Raumrichtungen nicht wesentlich ändern. Die Begriffe achsial und radial wechseln z.B. für eine Lasche eines Kettentriebs während dem Betrieb nie ihre Bedeutung, weil der ganze Mechanismus und alle Kettenelemente sich mehr oder weniger planar bewegen.

Vergleich zu 'Interaction Rule by Name'

Die hier beschriebene Funktion schliesst ein Objekt ggf. völlig und allgemein aus von entsprechenden Interaktionsberechnungen in bestimmte Raumrichtungen. Dies im Gegensatz zur Funktion *'Interaction Rule by Name'* welche die Begriffe 'axial' und radial' auch verwendet, aber nur in Zusammenhang mit einer bestimmten Gruppe von Objektpaarungen welche durch die Objektnamen festgelegt wird. Um auf den Zahnkettentrieb im letzten Abschnitt zurückzukommen, würde sich die hier beschriebene Funktion deshalb für eine Zahnlasche nicht eignen, weil diese axial mit den Nachbarlaschen und radial mit dem Kettenrad interagieren muss. Deshalb würde man in diesem Fall die Begriffe axial und radial besser mit den 'Interaction Rule by Name'-Regeln verwenden.

Interaktionsmethode (interaction method)

Syntax

```
SET PROPERTY (O1, INTERACT_METHOD, ELASTIC, double)
```

o1: Objekt Referenz {o1, o2, o3, ...}
INTERACT_METHOD, ELASTIC: reservierte Worte
double: Fliesskommawert [0..1]

Beschreibung

Bei der Interaktionsmethode geht es um das elastisch-plastische Verhalten der Oberfläche von Objekten beim Stoss. Auf die Stossenergie bezogen kann bei einem Stoss ein beliebiger Anteil der Energie absorbiert und der Rest wieder reflektiert werden. Mit dieser Funktion setzen wir diese Eigenschaft für ein einzelnes Objekt, d.h. wir setzen hier den Energieanteil des Stosses welcher reflektiert wird. Der Anteil wird als Wert im Bereich [0.0 ... 1.0] angegeben.
0: Nichts wird reflektiert -> rein plastischer Stoss.
1: Alles wird reflektiert -> rein elastischer Stoss
Standardwert = 0.1 -> Elastisch / Plastisch = 10% / 90%

Bei Kollisionen zwischen Objekten mit unterschiedlichen Einstellungen der Interaktionsmethode wird automatisch ein Zwischenwert berechnet.

Interaktionsart (interaction mode)

Syntax

```
SET PROPERTY (O1, INTERACT_MODE, {ACTIVE,PASSIVE,NO_INTERACTION})
```

o1: Objekt Referenz {o1, o2, o3, ...}
INTERACT_MODE, ACTIVE, PASSIVE, NO_INTERACTION: reservierte Worte

Beschreibung

Um die Leistung der Software zu steigern wird in sonar eine Klassifizierung der Objekte bezüglich ihrem allgemeinen Interaktionsverhalten verwendet. Sehen Sie dazu das entsprechende Kapitel im Tutorial, User Manual oder Anleitungsblatt. Default-mässig ist ein Objekt 'ACTIVE' und kümmert sich damit aktiv um Kollisionen mit anderen Objekten. In diesem Sinne müssen in Makros nur diejenigen Objekte mit dieser Funktion spezifiziert werden, welche einer anderen Regel gehorchen sollen.

Variablen (Primitives)

Viele Variablen der 'Primitives' können abgefragt und in Ausdrücken von 'sonar script' Kontrollsystemen verwendet werden. Deshalb finden die in diesem Kapitel beschriebenen Variablen hauptsächlich in 'sonar-SIM' ihre Anwendung, während die im letzten Kapitel beschriebenen Eigenschaften der 'Primitives' zur Hauptsache in sonar-LAB-Makros benutzt werden.
Die folgenden Variablen sind folglich sowohl zum Setzen als auch zum Abfragen von Werten gedacht. Diese Variablen können Bestandteil einer Formel sein, welche ein Resultatwert liefert. Andererseits kann dieser Resultatwert auch wieder einer Variablen zugewiesen werden. Allerdings sind nicht alle Variablen für beide Aufgaben geeignet. So macht es z.B. keinen Sinn einem Objekt eine Beschleunigung zu geben. Wenn schon, dann würde man auf das betreffende Objekt eine angemessene Kraft ausüben, welche diese Beschleunigung bewirkt. Auch das Setzen der Distanz oder der Kraft eines Links ist unsinnig. Ursache und Wirkung sollten nicht verwechselt werden. Wir unterscheiden zwischen verschiedenen Klassen von Variablen wie Geometrie, Physik, und allg. Variablen.

Ueberblick

Voraussetzungen

Die Komponenten in den folgenden Variablen beziehen sich immer auf das globale (|| lokale??) Koordinatensystem.

Geometrie

TABLE 4.

Variable	sonar script Bezeichnung	Kurzbezeichnung	Beispiel
Position	POSITION	POS	POS.X(O1), POS.Y(O12), POS.Z(O32)
Schwerpunkt	CENTER_MASS	CM	CM.X(O1), CM.Y(O12), CM.Z(O32)
Distanz, Abstand	DISTANCE	D	D.X(K1), D.Y(K12), D.Z(K32)

Physik

TABLE 5.

Variable	Variablen Bezeichnung	Kurzbezeichnung	Beispiel
Beschleunigung	ACCELERATION	A	A.X(O1), A.Y(O12), A.Z(O32)
Kraft	FORCE	F	F.X(O1), F.Y(O12), F.Z(O32)
Drehmoment	MOMENT_FORCE	M	M(O1)
externes Drehmoment	MOMENT_FORCE_EXT	M_EXT	M(O1)
Winkelbeschleunigung	ANGULAR_ACCELERATION	BETA	BETA(O1)
Winkelgeschwindigkeit	ANGULAR_VELOCITY	OMG	OMG(O1)
Impuls	MOMENTUM	P	P.X(O1), P.Y(O12), P.Z(O32)
Kin. Energie	E_KIN	E_KIN	E_KIN(O1)
Rotationsenergie	E_ROT	E_ROT	E_ROT(O1)
Kollisionskraft	FORCE_INTERACTION	F_IACT	F_IACT(O1)
Geschwindigkeit	VELOCITY	V	V.X(O1), V.Y(O12), V.Z(O32)

TABLE 5.

Variable	Variablen Bezeichnung	Kurzbezeichnung	Beispiel
externe Kraft	FORCE_EXT	F_EXT	F_EXT.X(O1), F_EXT.Y(O12), F_EXT.Z(O32)
Drehimpuls	ANGULAR_MOMENTUM	AM	AM(O1)
Widerstandsmoment	MOMENT_FRICTION	M_FRICTION	M_FRICTION(K1)
Zeit	TIME	T	T
Kin. Energie Total	E_KIN_TOT	E_KIN_TOT	E_KIN_TOT
Rot. Energie Total	E_ROT_TOT	E_ROT_TOT	E_ROT_TOT
Energie Total	E_TOT	E_TOT	E_TOT
Impuls Total	MOMENTUM_TOT	P_TOT	P_TOT

Allgemein

TABLE 6.

Variable	Variablen Bezeichnung	Kurzbezeichnung	Beispiel
Zyklus	CYCLE	C	C

Position

Syntax

```
POSITION.component(O1)
POS.component(O1)
```

POSITION, POS: reservierte Worte
component: {X|Y|Z} einen Koordinatenbezeichner des 3D-Positions-Vektors
o1: Objekt Referenz {o1, o2, o3, ...}

Beschreibung

Der Ausdruck ist eine Komponente {X|Y|Z} der globalen Position des Bezugspunktes, eines Objektes relativ zum Koordinatenursprung. Das folgende Beispiel berechnet den räumlichen Abstand zwischen den Bezugspunkten von zwei Objekten (O6, O4) und führt eine Reihe von Anweisungen durch, falls oder solange die beiden Objekte einen räumlichen Abstand von weniger als 12.5 cm voneinander haben.

Beispiel

```
DO IF (SQRT(SQR(POS.X(O6)-POS.X(O4))
           +SQR(POS.Y(O6)-POS.Y(O4))
           +SQR(POS.Z(O6)-POS.Z(O4))) < 12.5)
   statementList
END IF
```

Beachten Sie, dass die Linie mit dem Test `DO IF (...) auf einer einzigen Linie geschrieben werden muss. Sie ist hier nur zum besseren Verständnis umgebrochen worde.

Schwerpunkt

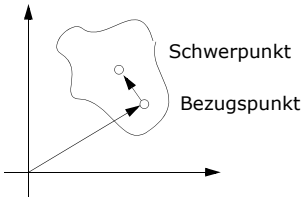
Syntax

```
CENTER_MASS.component(O1)
CM.component(O1)
```

CENTER_MASS, CM: reservierte Worte
component: {X|Y|Z} einen Koordinatenbezeichner des 3D-Positions-Vektors
o1: Objekt Referenz {o1, o2, o3, ...}

Beschreibung

Der Schwerpunkt ist die relative Position desselben zum Bezugspunkt des Objektes, ausgedrückt im lokalen Koordinatensystem des Objektes. Abgesehen von ein paar Ausnahmen, auf die an dieser Stelle nicht eingegangen werden soll, ist der Schwerpunkt allerdings mit dem Bezugspunkt identisch. Anders ausgedrückt, der Bezugspunkt fällt in den meisten Fällen mit dem Schwerpunkt zusammen. Folglich sind die Koordinaten des Schwerpunktes meistens Null.



Distanz, Abstand

Syntax

`DISTANCE.`*`component`*`(K1)`
`D.`*`component`*`(K1)`

DISTANCE, D: reservierte Worte
component: {X|Y|Z} einen Koordinatenbezeichner des 3D-Distanz-Vektors
K1: Link Referenz {k1, k2, k3, ...}

Beschreibung

Die absolute Auslenkung eines Links zwischen zwei Objekten. Dies ist die lokale Dehnung an der Verbindungsstelle. Betrachtet man den Link als Feder, dann entspricht dieser Abstand der Verlängerung der Feder unter Last. Ohne Last beträgt diese Distanz gleich Null. Je nachdem ob man für die Link-Konstante nur die lineare oder auch die quadratische Linkkonstante benutzt ist die Distanz eines Links eine homogene lineare Funktion der Last oder eben eine quadratische.

Beschleunigung

Syntax

`ACCELERATION.`*`component`*`(O1)`
`A.`*`component`*`(O1)`

ACCELERATION, A: reservierte Worte
component: {X|Y|Z} einen Koordinatenbezeichner des 3D-Beschleunigungs-Vektors
o1: Objekt Referenz {o1, o2, o3, ...}

Beschreibung

Die Funktion liefert die aktuelle Beschleunigung eines Objektes. Die Beschleunigungswerte können je nach Problemstellung mitunter stark verrauscht sein. Oft müssen die aufgezeichneten Beschleunigungswerte deshalb vor einer Verwendung in einer grafischen Darstellung geglättet werden. Im Prinzip wären entsprechende Messwerte auch in experimentellen Datenaufzeichnungen verrauscht, zumindest theoretisch. Allerdings sind der Aufzeichnungssensor selbst und die Aufzeichnungsapparatur durch ihre inneren Trägheiten bereits die ersten Filter welche diese Daten etwas dämpfen. Ein anderer Weg in den Besitz von Beschleunigungswerten zu gelangen, ist die Aufzeichnung von Geschwindigkeitswerten. Von diesen Daten wird später, zum Beispiel in EXCEL, die erste Ableitung gebildet was die Beschleunigungen ergibt.

Winkelbeschleunigung

Syntax

`ANGULAR_ACCELERATION.`*`component`*`(O1)`
`BETA.`*`component`*`(O1)`

ANGULAR_ACCELERATION, BETA: reservierte Worte
component: {X|Y|Z} einen Koordinatenbezeichner des 3D-Beschleunigungs-Vektors
o1: Objekt Referenz {o1, o2, o3, ...}

Beschreibung

Der Winkelbeschleunigungsvektor steht senkrecht auf der aktuellen Drehebene des Objektes. Die Anweisung liefert den vektoriellen Wert der im Moment wirkenden Winkelbeschleunigung am referenzierten Objekt.

Kraft

Syntax

```
FORCE.component(O1)
F.component(O1)
```

FORCE, F: reservierte Worte
component: {X|Y|Z} einen Koordinatenbezeichner des 3D-Kraft-Vektors
o1: Objekt Referenz {o1, o2, o3, ...}

Beschreibung

Wird mit dieser Anweisung eine Kraft auf ein Objekt berechnet und dem Objekt zugewiesen, dann wird diese Kraft zu den anderen wirkenden Kräften auf das Objekt, die allenfalls noch wirken, hinzuaddiert. Die Zuweisung ist nur komponentenweise möglich. Folglich können auf diese Weise einem Objekt mehrmals in Folge unterschiedlich berechnete Kräfte zugewiesen werden. Zugewiesene Kräfte mit dieser Variablen gelten in jedem Fall nur für den laufenden Berechnungszyklus. Nachdem die Summe aller Kräfte auf ein Objekt ausgewertet wurde, werden sie für den nächsten Rechenzyklus auf Null zurückgesetzt. Die Variable 'FORCE' funktioniert in diesem Sinne anders als die Variable 'FORCE_EXT', welche zwar auch immer additiv weiterverwendet wird, aber als Zahlenwert permanent stehen bleibt bis der Benutzer den Wert ändert oder ausschaltet.

Liefert die Anweisung ein Resultat dessen Wert in einem Term oder in einer Formel verwendet wird, dann berechnet die Anweisung 'FORCE' die vektorielle Summe aller Kräfte auf ein Objekt und setzt diesen Wert in der betreffenden Formel ein.

Kollisionskraft

Syntax

```
FORCE_INTERACTION.component(O1)
F_IACT.component(O1)
```

FORCE_INTERACTION, F_IACT: reservierte Worte
component: {X|Y|Z} einen Koordinatenbezeichner des 3D-Kraft-Vektors
o1: Objekt Referenz {o1, o2, o3, ...}

Beschreibung

Da in der sonar Software alle Stösse zwischen Objekten explizit nachgebildet werden, kann während einem Stoss auch der zeitliche Kraftverlauf an der Kontaktstelle der Objekte rechnerisch weiterverwendet werden. Es ist die wechselnde wirkende Kraft welche zeitlich über den gesamten Stoss integriert schliesslich den sog. Kraftstoss ergibt.

Drehmoment

Syntax

```
MOMENT_FORCE.component(O1)
M.component(O1)
```

MOMENT_FORCE, M: reservierte Worte
component: {X|Y|Z} einen Koordinatenbezeichner des 3D-Momenten-Vektors
o1: Objekt Referenz {o1, o2, o3, ...}

Das Drehmoment ist ein Vektor welcher mit seinem Fusspunkt senkrecht auf der Drehebene steht. Wie der Kraftvektor 'FORCE' kann das Drehmoment

gesetzt oder abgefragt und weiterverwendet werden. Die Zuweisung geschieht komponentenweise.

Externes Drehmoment

Syntax

```
MOMENT_FORCE_EXT.component(O1)
M_EXT.component(O1)
```

MOMENT_FORCE_EXT, M_EXT: reservierte Worte
component: {X|Y|Z} einen Koordinatenbezeichner des 3D-Momenten-Vektors
o1: Objekt Referenz {o1, o2, o3, ...}

Beschreibung

Das externe Drehmoment funktioniert wie das normale Drehmoment 'MOMENT_FORCE', mit dem Unterschied, dass das externe Drehmoment nach dem Setzen bis auf Weiteres stehen bleibt, während das normale Drehmoment nur für den laufenden Zyklus gilt.

Widerstandsmoment

Syntax

```
MOMENT_FRICTION(K1)
M_FRICTION(K1)
```

MOMENT_FRICTION, M_FRICTION: reservierte Worte
k1: Link Referenz {k1, k2, k3, ...}

Beschreibung

Werden Links als Lagerpunkte von Achsen verwendet, dann haben diese vorerst keine Lagerreibung. Mit dieser Anweisung wird das Widerstandsmoment eines einzelnen referenzierten Links gesetzt oder abgefragt. Damit lässt sich während einer Simulation eine Lagerreibung auch dynamisch verändern.

Geschwindigkeit

Syntax

```
VELOCITY.component(O1)
V.component(O1)
```

VELOCITY, V: reservierte Worte
component: {X|Y|Z} einen Koordinatenbezeichner des 3D-Geschwindigkeits-Vektors
o1: Objekt Referenz {o1, o2, o3, ...}

Beschreibung

Die Schwerpunktgeschwindigkeit des Objektes mit der Referenz 'O1' in globalen Koordinaten. Die Geschwindigkeit kann sowohl abgefragt und weiterverwendet als auch mit einer Zuweisung gesetzt werden.

Winkelgeschwindigkeit

Syntax

```
ANGULAR_VELOCITY.component(O1)
OMG.component(O1)
```

ANGULAR_VELOCITY, OMG: reservierte Worte
component: {X|Y|Z} einen Koordinatenbezeichner des 3D-Vektors
o1: Objekt Referenz {o1, o2, o3, ...}

Beschreibung

Die Komponenten des aktuellen Winkelgeschwindigkeitsvektors des Objektes mit der Referenz 'O1' in globalen(??) Koordinaten. Die Winkelgeschwindigkeit

kann sowohl abgefragt und weiterverwendet als auch mit einer Zuweisung gesetzt werden.

Impuls

Syntax

```
MOMENTUM.component(O1)
P.component(O1)
```

MOMENTUM, P: reservierte Worte
component: {X|Y|Z} einen Koordinatenbezeichner des 3D-Impuls-Vektors
o1: Objekt Referenz {o1, o2, o3, ...}

Beschreibung

Diese Anweisung liefert den translatorischen Impuls des referenzierten Objektes. Der Impuls wird dabei als (p = m * |v|) berechnet und ggf. weiterverwendet.

Gesamtimpuls

```
MOMENTUM_TOT
P_TOT
```

MOMENTUM_TOT, P_TOT: reservierte Worte

Beschreibung

Der Gesamtimpuls des gesamten Systems bzw. die Summe aller Einzelimpulse der Objekte.

Drehimpuls

Syntax

```
ANGULAR_MOMENTUM.component(O1)
AM.component(O1)
```

ANGULAR_MOMENTUM, AM: reservierte Worte
component: {X|Y|Z} einen Koordinatenbezeichner des 3D-Impuls-Vektors
o1: Objekt Referenz {o1, o2, o3, ...}

Beschreibung

Der Betrag des Eigendrehimpulses wird vom System als ($L = \Theta * \omega$) berechnet (??) und als skalare Grösse weiterverwendet.

Θ: Trägheitstensor des Objektes
ω: Winkelgeschwindigkeit

Energie

Syntax

```
E_KIN.component(O1)
E_ROT.component(O1)
E_KIN_TOT
E_ROT_TOT
E_TOT
```

E_KIN, E_ROT, E_KIN_TOT, E_ROT_TOT, E_TOT: reservierte Worte
o1: Objekt Referenz {o1, o2, o3, ...}

Beschreibung

Die Energie Anweisungen liefern die Energiewerte der Reihe nach wie folgt:

- die kinetische Energie des referenzierten Objektes

- die Rotationsenergie des referenzierten Objektes
- Die totale kinetische Energie sämtlicher Objekte im Modell
- die totale Rotationsenergie sämtlicher Objekte im Modell
- Die Gesamtenergie im Modell (`E_KIN_TOT + E_ROT_TOT`)

Man könnte diese Werte z.B. als Abbruchkriterium in einem Kontrollsystem benutzen, in welchem sich der Energiewert eines einzelnen Objektes ändert oder wo sich das Verhältnis der gesamten kinetischen zur gesamten Rotationsenergie verschiebt.

Zeit

Syntax

```
TIME
T
```

TIME, T: reservierte Worte

Beschreibung

Die Simulationszeit einer laufenden Simulation. Diese Zeit hat nichts zu tun mit der Weltzeit, wie sie ev. auf der Uhr am Handgelenk (oder dem Mobiltelefon) angezeigt wird. 'TIME' ist die Zeit in der Modellwelt Ihrer Simulation. Die Simulationszeit kann ggf. mit einem Menu-Befehl jederzeit auf Null zurückgesetzt werden, sogar während einer laufenden Simulation. Alle zeitabhängigen Variablen, Kontrollsysteme, grafischen Darstellungen und Einstellungen beziehen sich auf diese Zeit.

Zyklus

Syntax

```
CYCLE
C
```

CYCLE, C: reservierte Worte

Beschreibung

Die Anzahl durchgeführter Rechenzyklen. Die Zykluszahl wird jedesmal wenn die gesamte Physik des Modells einmal vollständig durchgerechnet wurde um Eins erhöht. Dieser Vorgang geschieht zusammen mit der Erhöhung der Simulationszeit um ein Zeitinkrement 'dt'. Die Anzahl Rechenzyklen pro Simulation liegt in der Regel im Bereich von 1 Million und 1 Milliarde. Auch die Zykluszahl kann wie die Simulationszeit 'TIME' jederzeit auf Null zurückgesetzt werden.

Trigger

(reserved parameter name for future use)

Operationen an Primitives

Dieses Kapitel behandelt die folgenden Operationen

- Primitives Selektieren
- Primitives Bewegen

Selektieren / Überblick

Der folgende Ueberblick zeigt die Möglichkeiten ein oder mehrere Objekte zu selektieren sowie die Selektion anderer Teile eines Modells.

Syntax (Objekte)

```
SELECT OBJECT (O1)
SELECT OBJECT (LAST_OBJECT)
SELECT OBJECT (LAST_OBJECT - n)
SELECT OBJECT (POINT, x1, y1, z1)
SELECT OBJECT (POINT, ALL, x1, y1, z1)
SELECT OBJECT (RECT, XY, x1, y1, x2, y2)
SELECT OBJECT (SPACE, x1, y1, z1, x2, y2, z2)
SELECT OBJECT (ALL)
SELECT OBJECT (COLOR_STD, index)
SELECT OBJECT (COLOR_RGB, red, green, blue)
```

Syntax (Rohdaten)

```
SELECT ELEMENT (E1)
SELECT CONTOUR (C1)
```

Syntax (Fixpunkte)

```
SELECT FIXPOINT (F1)
```

Syntax (Links)

```
SELECT LINK (K1)
SELECT LINK (x0, y0, z0)
```

o1, e1, c1, f1, k1: Referenzen auf entsprechende Objekt-Arten
LAST_OBJECT: das letzte Objekt in der Objektliste (Objekt Tool)
POINT: die nachfolgenden Variablen sind als Punkt zu interpretieren
ALL: die Operation bezieht sich auf alle Objekte, es werden alle Objekte selektiert.
XY: nimmt Bezug auf die X-Y-Hauptebene
RECT: ein Rechteck mit Blick auf eine Hauptebene
SPACE: ein Raumkörper (Quader)
n: ganze Zahl
x0, y0, z0; x1, y1, z1; x2, y2, z2: 3-dimensionale Punkte im kartesischen Koordinatensystem

Beschreibung

Die Anweisung ‘`SELECT OBJECT (POINT,x1,y1,z1)`’ selektiert das erste Objekt in der Objekt-Liste welches den Punkt `(x1,y1,z1)` in seinem Volumen enthält. Wird als zweiter Parameter zusätzlich noch ‘`ALL`’ angegeben, dann werden alle Objekte selektiert, welche diesen Punkt enthalten.

Der Parameter ‘`RECT`’ in der Hauptebene ‘`XY`’ bezieht sich auf die Front-Ansicht bzw. die X-Y-Ansicht. Alle Objekte welche in dieser Projektion liegen werden selektiert.

Der Parameter 'SPACE' schliesslich spannt einen Quader im 3-dimensionalen Raum auf und selektiert alle Objekte welche mit ihrem Schwerpunkt innerhalb dieses Quaders liegen.

Der Parameter 'index' muss sich im Bereich [0..25] befinden. Die drei Farbwerte (red, green, blue) sind ebenfalls ganzzahlige Werte im Bereich [0..255].

Die Anweisung 'SELECT LINK (x0,y0,z0)' selektiert den (punktförmigen) Link am angegebenen Ort im Modell.

eine weitere Referenzierungsart

Eine weitere Art bestimmte Objekte zu referenzieren, welche nicht im gleichen script erzeugt wurden, ist die folgende Methode.

Beispiel

```
SELECT OBJECT (POINT, x1, y1, z1) -- globale Koordinaten
SET VALUE (O1 = SELECTION)
SET PROPERTY (O1, DENSITY, 7.8)
```

Beschreibung

Dieses Beispiel erzeugt auf künstlichem Weg eine Referenz für ein bestimmtes, vorher selektiertes Objekt, indem man zuerst das Objekt an der globalen Position (x1, y1, z1) selektiert und anschliessend festlegt, diese Selektion habe nun den Namen bzw. die Referenzbezeichnung 'O1'. Im weiteren Verlauf kann dieses referenzierte Objekt 'O1' auf dem gleichen Weg weiter verwendet werden wie eine normal erzeugte Referenz.

Syntax

```
FIND OBJECT (O1, x1, y1, z1)
```

o1: Es wird eine Referenzierung auf 'O1' gesetzt
x1, y1, z1: kartesische Koordinaten eines globalen Raumpunktes.

Beschreibung

Eine ähnliche Funktion wie die letzte weiter oben weist das Objekt welches den angegebenen globalen Punkt (x1, y1, z1) enthält direkt der Objekt-Referenzierung 'O1' zu. Auch diese Objektreferenzierung kann anschliessend wie eine normale automatisch erzeugte Referenz weiterverwendet werden.

Bewegen (Translation) / Überblick

Syntax (Objekte)

```
TRANSLATE OBJECT(O1, ABSOLUTE, dx,dy,dz)
TRANSLATE OBJECT(O1, RELATIVE, dx,dy,dz)
TRANSLATE OBJECT(SELECTION, ABSOLUTE, dx,dy,dz)
TRANSLATE OBJECT(SELECTION, RELATIVE, dx,dy,dz)
```

Syntax (Objectgroup)

```
TRANSLATE OBJECTGROUP(O1, ABSOLUTE ,dx,dy,dz)
TRANSLATE OBJECTGROUP(O1, RELATIVE, dx,dy,dz)
TRANSLATE OBJECTGROUP(LAST_GROUP_NR,ABSOLUTE, dx,dy,dz)
TRANSLATE OBJECTGROUP(LAST_GROUP_NR,RELATIVE, dx,dy,dz)
```

Syntax (Objectsupergroup)

```
TRANSLATE OBJECTSUPERGROUP(O1, ABSOLUTE, dx,dy,dz)
TRANSLATE OBJECTSUPERGROUP(O1, RELATIVE, dx,dy,dz)
TRANSLATE OBJECTSUPERGROUP(LAST_SUPERGROUP_NR, ABS.., dx,dy,dz)
TRANSLATE OBJECTSUPERGROUP(LAST_SUPERGROUP_NR, REL.., dx,dy,dz)
```

Syntax (Grid)

```
TRANSLATE GRID (G1, ABSOLUTE, dx,dy,dz)
TRANSLATE GRID (G1, RELATIVE, dx,dy,dz)
```

```
TRANSLATE GRID (SELECTION, ABSOLUTE, dx,dy,dz)
TRANSLATE GRID (SELECTION, RELATIVE, dx,dy,dz)
```

o1: Objekt Referenz {o1, o2, o3, ...}
ABSOLUTE: die Koordinaten sollen als absolute globale Koordinaten interpretiert werden
RELATIVE: die Koordinaten sollen relativ zur aktuellen Position der Objekte interpretiert werden
SELECTION, LAST_GROUP_NR, LAST_SUPERGROUP_NR: reservierte Worte
dx, dy, dz: 3-dimensionale kartesische Koordinatenwerte des Verschiebungsvektors

Beschreibung

Die betreffenden Objekte werden dreidimensional und translatorisch, d.h. ohne Drehungen, im Raum verschoben. Die Verschiebungen können immer absolut zum Ursprung oder relativ zur aktuellen Position erfolgen. Diese Operationen imitieren im Prinzip das Verschieben von Objekten in den Ansichten mit der Maus

Bewegen (Rotation)

Syntax (Objects)

```
ROTATE OBJECT (O1, cx,cy,cz, wx,wy,wz)
ROTATE OBJECT (SELECTION, cx,cy,cz, wx,wy,wz)
```

Syntax (Objectgroup)

```
ROTATE OBJECTGROUP(O1, cx,cy,cz, wx,wy,wz)
ROTATE OBJECTGROUP(LAST_GROUP_NR, cx,cy,cz, wx,wy,wz)
```

Syntax (Objectsupergroup)

```
ROTATE OBJECTSUPERGROUP(O1, cx,cy,cz, wx,wy,wz)
ROTATE OBJECTSUPERGROUP(LAST_SUPERGROUP_NR, cx,cy,cz, wx,wy,wz)
```

Syntax (Grid)

```
ROTATE GRID (G1, cx, cy, cz, wx, wy, wz)
```

o1, g1: : Referenzen auf entsprechende Objekt-Arten
SELECTION, LAST_GROUP_NR, LAST_SUPERGROUP_NR: reservierte Worte
cx, cy, cz: 3-dimensionale kartesische Koordinatenwerte des Drehzentrums
wx, wy, wz: Drehwinkel für die drei Drehachsen in [rad].

Beschreibung

Die betroffenen Objekte werden mit den angegebenen Winkeln (wx, wy, wz) in [rad] um den Punkt (cx, cy, cz) als Drehzentrum gedreht. Es empfiehlt sich immer nur einen der drei Winkel (wx, wy, wz) zu verwenden und die beiden anderen auf Null zu setzen. Durch mehrmalige Anwendung dieser Anweisung können die Objekte auf diesem Weg genau in der gewünschten Reihenfolge um das Zentrum gedreht werden. Sind zwei oder drei Werte für (wx, wy, wz) von Null verschieden, dann führt das System diese Drehungen in einer vorgegebenen Reihenfolge durch. Und zwar der Reihe nach: wz -> wx -> wy. Da Drehungen nicht kommutativ sind ergibt im Allgemeinen jede Reihenfolge ein anderes Ergebnis.

Bewegen (mit Objektmatrix)

Syntax (Rohdaten)

```
MOVE ELEMENT (E1, MOVE_MATRIX, O1)
MOVE ELEMENT (E1, MOVE_MATRIX, SELECTION)
MOVE ELEMENT (SELECTION, MOVE_MATRIX, O1)
MOVE ELEMENT (SELECTION, MOVE_MATRIX, SELECTION)
```

e1, o1: : Referenzen auf entsprechende Objekt-Arten
MOVE_MATRIX, SELECTION, : reservierte Worte

Beschreibung

Statt eine Drehung direkt durch die Angabe eines Winkels und einer Achse zu definieren, so wie im letzten Abschnitt, kann ein Rohdaten-Element in einer Anweisung mit der Drehmatrix eines bereits bekannten Objektes gedreht

und translatorisch verschoben werden, welches entweder referenziert oder selektiert wird. In der letzten der vier Anweisungen (oben) müssen also vorab zwei Selektionen gesetzt werden, die Selektion eines Elementes und eines Objektes. Der erste Parameter bezieht sich immer auf ein Element, der letzte Parameter immer auf ein Objekt.
Die Elemente werden mit dieser Anweisung simultan mit einem Objekt bewegt. Und zwar rotativ als auch translatorisch. Die Elemente machen im dreidimensionalen Raum dieselbe Bewegung wie das betreffende Objekt ausgehend von seiner Nullposition bis zur aktuellen Position und Drehung im Raum. Das System benutzt dazu die gespeicherte Rotationsmatrix und die Koordinaten des Bezugspunktes des betreffenden Objektes.

Kombinierte Bewegung

Syntax

```
MOVE OBJECT (O1, ABSOLUTE, x0, y0, z0, wx, wy, wz)
MOVE OBJECT (O1, RELATIVE, x0, y0, z0, wx, wy, wz)
MOVE OBJECT (SELECTION, ABSOLUTE, x0, y0, z0, wx, wy, wz)
MOVE OBJECT (SELECTION, RELATIVE, x0, y0, z0, wx, wy, wz)
```

o1: : Referenz auf Objekt
ABSOLUTE, RELATIVE, SELECTION: reservierte Worte
x0, y0, z0: absolute kartesische Koordinaten oder Verschiebungsvektor (je nach Param.2)
wx, wy, wz: Drehungswinkel in [rad] um die einzelnen Koordinatenachsen

Beschreibung

Die Befehle kombinieren alle Rotationen und die Translation in eine einzige Anweisung. Als Erstes werden die Rotationen durchgeführt und in einem zweiten Schritt die Translation. Die Rotationen werden in einer festen Reihenfolge abgearbeitet: Rot(Z) -> Rot(X) -> Rot(Y). Deshalb sind die Funktionen etwas unübersichtlich und auch nur eingeschränkt anwendbar. Wir empfehlen aus diesem Grund besser die elementaren Funktionen '`ROTATE OBJECT`' und '`TRANSLATE OBJECT`' mehrmals in der gewünschten Reihenfolge anzuwenden.

Löschen

Makro

```
CLEAR OBJECT (O1 || SELECTION)
```

O1: Element Referenz {o1, o2, o3, ...}
SELECTION: (reserviertes Wort) -> es müssen Objekte selektiert sein.

Beschreibung

Wir verwenden hier in 'sonar script' ausdrücklich das Kommando '`CLEAR`' und nicht '`DELETE`' um Objekte zu Löschen. Das bedeutet, dass die gelöschten Objekte nicht in die Zwischenablage übertragen und folglich auch nicht mit '`PASTE`' wieder eingesetzt werden können. Das Löschen und wieder Einsetzen von Objekten innerhalb eines scripts macht letztlich keinen Sinn und ist nicht notwendig bzw. kann ggf. besser anders gelöst werden (sehen Sie dazu '`DUPLICATE`')

Makro

```
CLEAR ALL
```

Beschreibung

Diese Anweisung löscht alle was selektiert ist, wie auch immer sich die Selektion zusammensetzt. Das können nur Objekte oder eine Kombination aus Elementen und Objekten sein.

Duplizieren

Syntax (in place)

```
DUPLICATE OBJECT (O1,        INPLACE || TRUE)
DUPLICATE OBJECT (SELECTION, INPLACE || TRUE)
DUPLICATE OBJECT (ALL,       INPLACE || TRUE)
```

Syntax (displaced)

```
DUPLICATE OBJECT (O1,        FALSE)
DUPLICATE OBJECT (SELECTION, FALSE)
DUPLICATE OBJECT (ALL,       FALSE)
```

O1: Element Referenz {o1, o2, o3, ...}
SELECTION: (reserviertes Wort) -> es müssen Objekte selektiert sein
ALL: (reserviertes Wort) -> Alle Objekte des Modells werden dupliziert
INPLACE || TRUE: Alle duplizierten Objekte befinden exakt an der gleichen Position
FALSE: Alle duplizierten Objekte werden räumlich verschoben

Beschreibung

Durch Objekt-Referenzen oder Selektionen bezeichnete Objekte werden wahlweise an derselben Position oder seitlich verschoben dupliziert. Der Vorgang des Duplizierens kopiert nicht nur das äussere Erscheinungsbild des ursprünglichen Objektes sondern auch seine allgemeinen und physikalischen Eigenschaften. Das neue Objekte befindet sich immer am Ende der Liste des ObjectTool's und kann mit der Variablen '`LAST_OBJECT`' ggf. referenziert werden.

Links

Erzeugen

Links sind punktförmige elastische Verbindungen zwischen Objekten. Links können unter Spannung ein elastisch-plastisches Verhalten haben. Sehen Sie dazu 'Gruppeneigenschaften / Materialmodell'

Syntax Normaler Link

```
CREATE LINK (K1, NORMAL, O1, O2, x0,y0,z0)
CREATE LINK (K1, NORMAL, O1, O2, E1)
CREATE LINK (K1, NORMAL, O1, F1)
```

k1: erzeugte Link Referenz
NORMAL, CTR120: reservierte Worte
o1, o2: die beiden zu verbindenden Objekte
x0, y0, z0: globale kartesische Raumkoordinaten
e1: statt explizit die Koordinaten (x0, y0, z0) zu übergeben, kann auch ein bereits erzeugtes Element 'E1' vom Typ 'POINT' den Ort des Links spezifizieren.
f1: eines der beiden Objekte kann auch ein Fixpunkt sein. Die Angabe der Position erübrigt sich in diesem Fall.

Beschreibung

Ein normaler Link, erzeugt mit dem Parameter 'NORMAL', ist vergleichbar mit einer Zug- oder Druckfeder. Ein normaler Link reagiert in diesem Sinn auf Längenänderungen. Ein normaler Link hat im Moment der Erzeugung die Länge Null. Entstehen im Laufe der Simulation Verschiebungen und damit Spannungen zwischen zwei verlinkten Objekten, dann bekommt der Link eine endliche Länge. Zusammen mit den Link-Konstanten und weiteren Einflussgrössen wird daraus die Linkkraft berechnet, welche der Auslenkung des Links entgegenwirkt. Sowohl die Steifigkeit als ggf. andere Eigenschaften des Links werden mit separaten Anweisungen gesetzt. Die obigen Anweisungen versehen den neuen Link vorerst mit den sog. Default-Link-Eigenschaften. Deshalb folgen in einem sonar script einem neuen Link in der Regel weitere Anweisungen welche die spezifischen Eigenschaften des Links setzen.

Syntax Biege- und Torsions-Link

```
CREATE LINK (K1, BENDING, O1, O2, x0,y0,z0)
CREATE LINK (K1, TORSION, O1, O2, x0,y0,z0)
```

k1: erzeugte Link Referenz
BENDING, TORSION: reservierte Wörter
o1, o2: Referenzen auf vorhandene Objekte
x0, y0, z0: globale kartesische Raumkoordinaten

Beschreibung

Biege- und Torsions-Links sind zusätzliche Links, die am Ort eines bereits gesetzten normalen Links angebracht werden können. Diese beiden zusätzlichen Links halten die beiden Objekte aber nicht wirklich zusammen, so wie das ein normaler Link kann, sondern üben nur noch zusätzliche Kräfte aus. Aus diesem Grund müssen Biege- und/oder Torsions-Links immer zusammen mit normalen Links verwendet werden. Eine Link-Kombination bestehend aus allen drei Linksorten kontrolliert schliesslich alle möglichen Bewegungen welche zwischen zwei Objekten auftreten können. Das folgende Script erzeugt zwei Zylinder und verbindet sie mit allen drei Link-Arten

Beispiel

```
BEGIN SCRIPT Verbindung
-- Erzeuge 2 Zylinder und verbinde sie mit 3 Link-Arten
```

```
CREATE OBJECT  (O1, CYLINDER, 0, 0, 0.0, 0, 0, 6.0,  0.5)
CREATE OBJECT  (O2, CYLINDER, 0, 0, 6.0, 0, 0, 12.0, 0.5)
CREATE LINK (K1, NORMAL,  O1, O2, 0, 0, 6.0)
CREATE LINK (K2, BENDING, O1, O2, 0, 0, 6.0)
CREATE LINK (K3, TORSION, O1, O2, 0, 0, 6.0)
-- end of script
```

weitere Erzeugungsmethoden

Syntax

```
CREATE LINK (K1, NORMAL, SELECTION)
```

k1: erzeugte Link Referenz
NORMAL, SELECTION: reservierte Wörter

Beschreibung

Anstelle dass zur Erzeugung eines Links entsprechende Objektreferenzen und Koordinaten angegeben werden, kann vorab eine Selektion von gleichwertigen Information selektiert werden. Konkret müsste man in diesem Fall zwei verschiedene Objekte und einen Rohdatenpunkt selektieren, bevor diese Anweisung aufgerufen wird. Diese Selektion muss natürlich auch im gleichen Script mit den entsprechenden Selektions-Anweisungen durchgeführt werden. Allerdings sind auf diesem Weg auch Selektionen von Objekten möglich, welche in einem anderen Script erzeugt wurden (siehe Primitives/Selektieren).

Syntax

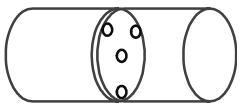
```
CREATE LINK (K1, NORMAL, CTR120, O1, O2)
CREATE LINK (K1, NORMAL, CTR120, O1, O2, mode)
```

k1: erzeugte Link Referenz
o1, o2: Referenzen auf vorhandene Objekte
NORMAL: reserviertes Wort
CTR120: reserviertes Wort. Es sollen 4 Links gesetzt werden, einer im Zentrum und 3 an der Peripherie.
mode: {STRAIGHT, DEFLECTED}
STRAIGHT: zero angle in the straight position between two objects.
DEFLECTED: zero angle in the actual and relativ object position (as it is)

Beschreibung

Um zwei Objekte, wie z.B. zwei einzelne Zylinder eines biegsamen Stabes, elastisch und formfest zu verbinden, gibt es die folgenden Methoden:

- Normal Link + Biege-Link + Torsions-Link -> sog. NBT-Link
- 3 normale periphere Links (3 x 120°)
- 1 normaler Zentral-Link + 3 normale periphere Links -> sog. CTR120-Link



Der sog. CTR120-Link ist bezüglich der Einstellung der physikalischen Eigenschaften flexibler als die einfachere Kombination mit nur drei peripheren Links. In einem CTR120-Link können die peripheren Links so eingestellt werden, dass sie unter der Annahme eines zentralen Scharniers die Biege- oder Torsions-Steifigkeit optimal wiedergeben. Der zentrale Link kann anschliessend unabhängig davon so bemessen werden, dass er mit den peripheren Links zusammen den E-Modul korrekt berechnet. Die genaue Einstellung der einzelnen Links wird nicht mit dieser Anweisung durchgeführt sondern anschliessend mittels der Funktion '`SET VALUE (E_MODUL = double, ...)`' festgelegt. Diese Funktion berechnet dann unter Berücksichtigung der Dimensionen der einzelnen Objekte einer Gruppe die einzelnen Link-Konstanten der CTR120-Links.

Beim einfacheren Kombinationslink mit lediglich drei peripheren Links übernimmt jeder Link 1/3 des Emoduls. Anschliessend werden die drei Links radial soweit vom Zentrum angeordnet, dass sie mit den gegebenen Link-

Konstanten und Objektdimensionen auch die Biegesteifigkeit korrekt wiedergeben.

Die Anweisung ist auf bestimmte Objektkombinationen beschränkt. Folgende Paarungen sind bisher zugelassen:

- Zylinder - Zylinder
- Zylinder - Torus-Segment
- Zylinder - Kugel

Syntax

```
CREATE LINK (K1, NORMAL, AUTOMATIC, E1)
CREATE LINK (K1, NORMAL, AUTOMATIC, E1, O1)
CREATE LINK (K1, NORMAL, AUTOMATIC, CTR120, E1)
```

k1: Link Referenz auf den neuen Link
e1, o1: Referenzen auf entsprechende Objekt-Arten
NORMAL, AUTOMATIC, CTR120: reservierte Worte

Beschreibung

Der Parameter '`AUTOMATIC`' will uns sagen, dass die Verbindung am Ort der Position von Element '`E1`', geschehen soll, die betroffenen Objekte aber von der Funktion selbst gefunden werden sollen.

In der zweiten Anweisung wird ein zusätzliches Objekt '`O1`' referenziert. In diesem Fall soll folglich nur noch das zweite, dazu passende, Objekt gefunden werden.

Im letzten Fall sollen am Ort der Position von Element 'E1' insgesamt vier Links gesetzt werden, welche zusammen einen sog. CTR120-Link bilden (sehen Sie dazu '`CREATE LINK (K1, NORMAL, CTR120, O1, O2)`' in diesem Kapitel)

Ganze Objektgruppen Linken

Syntax

```
ATTACH OBJGRP_OBJGRP(O1, O2, mode)
```

o1, o2: Referenzen auf vorhandene Objekte
mode: NORMAL, NBT, CTR120, 3x120: reservierte Parameter

Beschreibung

Beide Objekt-Referenzen (O1, O2) sind Repräsentanten einer Objektgruppe. Es werden allerdings nicht alle Objekte der einen Gruppe mit allen Objekten der anderen Gruppe verlinkt. Vielmehr überprüft das System. welche Objektpaarungen sinnvoll zusammenpassen und verlinkt nur diese. Die Kombination der Objekte geschieht dabei nach der Methode, wo jedes Objekt der einen Gruppe überprüft, ob es ein Objekt in der anderen Gruppe gibt, welches mit dem ersten überlappt oder dieses berührt.
(ev. gehört diese Funktion zu einem Modul)

Link-Konstante setzen

Syntax

```
SET PROPERTY(K1, C_LINK, value)
```

k1: Link Referenz
C_LINK: reserviertes Wort
value: Link Konstante (Fliesskommawert)

Beschreibung

Dies ist die meistgebrauchte Standardfunktion zum Setzen von Link-Konstanten. Man benötigt sie praktisch in allen scripts wo Links gesetzt werden. Sie wird auf einen einzelnen Link angewendet welcher mit seiner

Referenzbezeichnung angegeben wird. Die Anweisung ist auf alle Link Typen anwendbar.

Syntax

```
SET VALUE (C_LINK = double, linktype, OBJECTNAME, "objectname")
```

C_LINK, OBJECTNAME: reservierte Worte
linktype: {NORMAL, BENDING, TORSION}

Beschreibung

Allen Links eines bestimmten Typs in Verbindung mit Objekten eines gewissen Namens wird ein neuer Wert für die betreffende Link-Konstante zugewiesen. Es gibt zwar ähnliche Funktionen für Objektgruppen, aber hier wählen wir nicht eine bestimmte Gruppe von Objekten aus, sondern alle mit einem bestimmten Namen, der natürlich vorher durchgehend gesetzt worden sein muss.

Beispiel

```
SET VALUE (C_LINK = 0.1, BENDING, OBJECTNAME, "Drahtelement")
```

Beachten Sie, dass die 'Gänsefüsschen' beim Objektnamen auch gesetzt werden müssen.

Syntax

```
SET GROUP_PROPERTY(O1, C_LINK, CTR120, K_ctr, K_r)
```

k1: Link Referenz
C_LINK: reserviertes Wort
K_ctr, K_r: Link Konstanten (Fliesskommawerte)

Beschreibung

Diese Anweisung bezieht sich auf alle Links einer Objektgruppe welche mit der Link-Kombinationen des Typs 'CTR120' verbunden sind. Ein CTR120-Link besteht aus insgesamt vier Links (1 Zentral-Link und 3 periphere Links). Referenziert werden diese Links mit einem Objekt 'O1' der Objektgruppe. Die beiden letzten Parameter 'K_ctr' und 'K_r' sind die eigentlichen Link-Konstanten und zwar zuerst die Link-Konstante für den zentralen Link und anschliessend diejenige für die drei peripheren Links. Mit dieser Anweisung werden die Link-Konstanten der CTR12-Links nicht automatisch berechnet sondern vom Benutzer explizit vorgegeben.

Eigenschaften setzen

Syntax

```
SET PROPERTY(K1, C_DAMPING, double)
SET PROPERTY(K1, ANGLE, {X|Y|Z}, double)  -- K1 = Torsions Link
SET PROPERTY(K1, MOMENT_FRICTION, double) -- K1 = normaler Link
SET PROPERTY(K1, C_LINK, CTR120, Kc, Kr) -- K1 = 1 von 4 Links
```

K1: Link Referenz des neuen Links
ANGLE, C_DAMPING. C_LINK, MOMENT_FRICTION: reservierte Worte
{X|Y|Z}: Es wird eine Koordinatenrichtung gesetzt (X oder Y oder Z)
double: Fliesskommawert
Kc: central Link; Kr: lateral Link

Beschreibung

Die erste Anweisung setzt die Dämpfungskonstante eines bestimmten Links welcher mit 'K1' referenziert ist. Auch diese Anweisung ist für alle Link-Typen einsetzbar.

Mit dem Parameter '`ANGLE`' sprechen wir in der zweiten Anweisung den Winkel eines Torsions-Links an, welcher in der aktuellen Drehlage der beiden Objekte die er verbindet einem bestimmten Drehwinkel entsprechen soll. Die beiden Objekte hätten dann zum Zeitpunkt des Simulationsstarts bereits ein

Drehmoment zueinander. Die Funktion kann nur für die drei Hauptachsenrichtungen benutzt werden

Die dritte Anweisung setzt eine Lagerreibung. Bei jeder Drehbewegung wirkt auf die beiden Objekte, die durch den (normalen) Link 'K1' verbunden sind, ein Widerstandsmoment welches der aktuellen relativen Drehrichtung entgegenwirkt. Ist die relative Drehbewegung zwischen den beiden Objekten gleich Null, dann ist auch das Widerstandsmoment gleich Null.

Die letzte Anweisung berifft einen einzelnen CTR120 Link.

Biegefestigkeit

Syntax

```
SET VALUE (BENDING_STRENGTH = value, GROUP_NR,
                                     LAST_GROUP_NR, f)
SET VALUE (BENDING_STRENGTH = value, SUPERGROUP_NR,
                                      LAST_SUPERGROUP_NR, f)
```

BENDING_STRENGTH, GROUP_NR, LAST_GROUP_NR, SUPERGROUP_NR. LAST_SUPERGROUP_NR: reservierte Worte
f: reduction or multiplication factor: mehrere Biege-Links zwischen 2 Objekten oder ein Hüllzylinder repräsentiert mehrere Drähte.
value: Fliesskommawert

Beschreibung

Diese Anweisung richtet sich ausschliesslich an sog. Bending-Links und nur diese werden berücksichtigt. Im Uebrigen ist diese Funktion bislang nur für Zylinder-Verbindungen implementiert. Es wird das maximale Biegemoment zwischen zwei Zylindern festgelegt. Damit wird mit den gegebenen Link-Konstanten der Bending-Links indirekt auch ein maximaler Biegewinkel vorgegeben. Wird während einer Simulation der Biegewinkel zwischen den betreffenden Objekten grösser als der Maximalwert, dann wird das Biegemoment jeweils auf den definierten Maximalwert zurückgesetzt.

Mit der nächsten Anweisung kann dieser maximale Grenzwert für das Biegemoment ein- oder ausgeschaltet werden.

Syntax

```
SET STATE (BENDING_STRENGTH, ON || OFF, GROUP_NR,
                                        LAST_GROUP_NR)
SET STATE (BENDING_STRENGTH, ON || OFF, SUPERGROUP_NR,
                                        LAST_SUPERGROUP_NR)
```

BENDING_STRENGTH, GROUP_NR, LAST_GROUP_NR, SUPERGROUP_NR. LAST_SUPERGROUP_NR: reservierte Worte
ON oder OFF: bool'sche Werte (reservierte Worte)

Beispiel

Das folgende Beispiel erzeugt einen Draht bestehend aus einer Kette von zylindrischen Objekten und verbindet die Zylinder paarweise. Der letzte erzeugte Zylinder wird jeweils mit dem vorangehenden mit drei Links verbunden (NBT-Link).

```
BEGIN SCRIPT Draht
-- Drahtdurchmesser = 1.0 mm
-- Drahtlänge = 60.0 mm
LOOP FOR (I, 1, 20, 1)
   CREATE OBJECT (O1,CYLINDER,0, 0, (I-1)*0.6, 0, 0, I*0.6, 0.05)
```

```
   DO IF (I>1)
      CREATE ELEMENT (E1, POINT, 0, 0, (I-1)*0.6) -- Link-Punkt

      DESELECT ALL
      SELECT ELEMENT (E1)
      SELECT OBJECT (LAST_OBJECT - 1)
      SELECT OBJECT (LAST_OBJECT)

      CREATE LINK (K1, NORMAL, SELECTION) -- NBT Link
      CREATE LINK (K2, BENDING, SELECTION)
      CREATE LINK (K3, TORSION, SELECTION)

      SET PROPERTY(K1, C_LINK, 0.1)
      SET PROPERTY(K2, C_LINK, 0.02)
      SET PROPERTY(K3, C_LINK, 0.008)
   END IF
END FOR
-- end of script
```

Fixpunkte

Fixpunkte sind masselose, unbewegliche Ankerpunkte im Raum. Fixpunkte haben keine physikalischen Eigenschaften. Ihre Aufgabe besteht einzig darin, mittels Links gewisse Objekte daran befestigen zu können. Daraus ist ersichtlich, dass das befestigte Objekt selbst elastisch am Fixpunkt befestigt ist. Das befestigte Objekt kann sich im Rahmen der Elastizität des Links noch um den Fixpunkt herum bewegen. Deshalb ist es auch möglich die wirkenden Kräfte zwischen Objekt und Fixpunkt aufzuzeichnen oder anderweitig zu verwenden. Fixpunkte ohne angedockte Objekte haben keine Funktion und sind überflüssig.

Erzeugen

Makro

```
CREATE FIXPOINT (F1, x0, y0, z0)
CREATE FIXPOINT (F1, SELECTION)
```

F1: Fixpunkt Referenz {f1, f2, f3, ...}
x0, y0, z0: kartesiche Koordinaten des Fixpunktes
SELECTION: (reserviertes Wort) -> ein Rohdatenpunkt muss selektiert sein.

Beschreibung

Die Position des zu erzeugenden Fixpunktes kann entweder explizit mit den Koordinaten angegeben oder durch eine vorangehende Selektion eines Punktes vermittelt werden. Die Anweisung erzeugt eine Referenzbezeichnung welche bei der Erzeugung eines Links zwischen Objekt und Fixpunkt verwendet werden kann.

Selektieren

Makro

```
SELECT FIXPOINT (F1)
```

F1: Fixpunkt Referenz {f1, f2, f3, ...}

Beschreibung

Die Anweisung kann dazu benutzt werden um einen Links zwischen selektierten Objekten zu setzen, wobei eines der selektierten Objekte ein Fixpunkt sein kann.

Links an Fixpunkten

Makro 1

```
CREATE LINK (K1, NORMAL, O1, F1)
```

K1: Link Referenz {k1, k2, k3, ...}
NORMAL; reserviertes Wort
O1, F1: Objekt- und Fixpunkt-Referenz zwischen denen der Link gesetzt werden soll.

Makro 2

```
CREATE LINK (K1, NORMAL, SELECTION)
```

K1: Link Referenz {k1, k2, k3, ...}
NORMAL; reserviertes Wort
SELECTION: reserviertes Wort

Beschreibung

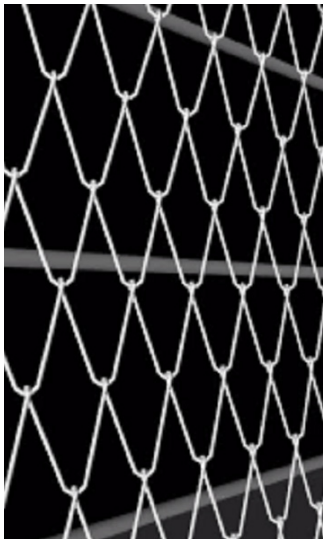
In ‘Makro 2’ darf höchstens eines der beiden selektierten Objekte ein Fixpunkt sein. Die Reihenfolge der Selektion spielt keine Rolle. Die Link-Eigenschaften werden wie üblich separat mit dem Kommando ‘`SET PROPERTY (K1, ...)`’ festgelegt.

Gruppeneigenschaften

Ueberblick

Wir unterscheiden im Moment zwei Stufen von Gruppen. Es sind dies die normale Gruppe und die Supergruppe. Die Supergruppe ist dazu gedacht Gruppen von Gruppen zu bilden. Allerdings ist dies keine Voraussetzung. Ein Objekt kann grundsätzlich auch einer Supergruppe angehören ohne Mitglied einer untergeordneten Gruppe zu sein. Im Prinzip handelt es sich bei dieser Terminologie einfach um zwei unterschiedliche Gruppennummern die ein Objekt haben kann. Aber die Idee dahinter ist letztlich schon die, dass der Benutzer diese Gruppenbezeichnungen in einer hierarchischen Art und Weise einsetzt.

Beispiel



Ein typisches Beispiel für die Verwendung dieser beiden Gruppen wäre ein Maschengitter wie nebenstehend abgebildet. Man würde den einzelnen Drähten, welche aus einer grösseren Anzahl von Zylinderelementen bestehen, jeweils eine individuelle Gruppennummer geben. Allen Drähten des gesamten Gitters zusammen würde man zusätzlich eine gemeinsame Supergruppennummer geben. Jedes Zylinderelement des Gittermodells hätte so eine Supergruppennummer, welche das Objekt als Mitglied eines bestimmten Gitters charakterisiert und zusätzlich eine Gruppennummer welche angibt, zu welchem Draht in diesem Gitter das Objekt gehört.

Der Sinn für diese Bezeichnungen ist letztlich, für die Vergabe von Eigenschaften gezielt auf die Objekte zugreifen zu können und gewisse Eigenschaften gleichsam für die ganze Gruppe oder Supergruppe zu setzen. Das Vergeben von Gruppennummern ist grundsätzlich nicht zwingend, aber empfohlen. Man sollte sich bei der Organisation der Objekte in Gruppen immer die Ueberlegung machen, welche Objekte man eventuell später zum Setzen oder Aendern von Eigenschaften als Gruppe ansprechen möchte.

Gruppennummer Erzeugen

Syntax

```
SET VALUE (NEW_GROUP_NR)
SET VALUE (NEW_SUPERGROUP_NR)
```

NEW_GROUP_NR, NEW_SUPERGROUP_NR: reservierte Worte

Beschreibung

Diese beiden Anweisungen erzeugen eine neue Gruppen- resp. Supergruppen-Nummer. Dies geschieht, indem das System die höchste existierende Nummer im Modell ermittelt, diese um Eins erhöht und intern als neue letzte Gruppen- bzw. Supergruppen-Nummer speichert. Diese aktuellen letzten Gruppennummern können jederzeit mit den Variablen ‘`LAST_GROUP_NR`’ und ‘`LAST_SUPERGROUP_NR`’ abgefragt bzw. als Parameter weiterverwendet werden. Diese Nummern können in der Folge beliebig vielen Objekten zugewiesen werden. Normalerweise geschieht das in einem ‘sonar script’ innerhalb von Schleifen (Loops).

Einzelne Objekte zu Gruppe hinzufügen

Syntax

```
SET PROPERTY (O1, GROUP_NR, LAST_GROUP_NR)                    (*)
SET PROPERTY (O1, SUPERGROUP_NR, LAST_SUPERGROUP_NR)          (*)
```

O1: Objekt Referenz
(*) O1 ist hier ein bestimmtes einzelnes Objekt
GROUP_NR, SUPERGROUP_NR: reservierte Worte
LAST_GROUP_NR, LAST_SUPERGROUP_NR: ganze Zahl (die letzten erzeugten Gruppennummern)

Beschreibung

Einem einzelnen referenzierten Objekt 'O1' wird die letzte erzeugte Gruppen- resp. Supergruppen-Nummer zugesprochen. Der zweite Parameter (`GROUP_NR, SUPERGROUP_NR`) gibt an, welche Sorte von Gruppennummer in diesem Zusammenhang gemeint ist bzw. gesetzt werden soll. In den Objekteigenschaften des betreffenden Objektes wird fortan angezeigt, dass dieses Objekt die betreffende Gruppennummer besitzt.

Gruppe zu Supergruppe hinzufügen

Syntax

```
SET GROUP_PROPERTY (O1, SUPERGROUP_NR, LAST_SUPERGROUP_NR)   (**)
SET GROUP_PROPERTY (LAST_GROUP_NR, SUPERGROUP_NR,
                                    LAST_SUPERGROUP_NR)
```

O1: Objekt Referenz
(**) O1 ist hier ein Repräsentant einer Objektgruppe
SUPERGROUP_NR: reservierte Worte
LAST_GROUP_NR, LAST_SUPERGROUP_NR: ganze Zahl (die letzten erzeugten Gruppennummern)

Beschreibung

Die erste Anweisung heisst in Worten: Die Objektgruppe, repräsentiert mit der Objektreferenz 'O1', soll der Supergruppe mit der Nummer 'letzte Supergruppe' hinzugefügt werden.

Analog hat die zweite Anweisung die Bedeutung: Die Objektgruppe mit der Nummer 'letzte Objektgruppe' soll der Supergruppe mit der Nummer 'letzte Supergruppe' hinzugefügt werden.

Beachten Sie, dass hier die Anweisung '`SET GROUP_PROPERTY`' benutzt wird, was darauf hindeutet, dass eine Gruppeneigenschaft gesetzt wird. Deshalb hat der erste Parameter 'O1' hier die Funktion eines Repräsentanten einer Objektgruppe.

Gruppeneigenschaften benutzen

Die meisten Eigenschaften können in 'sonar script' wahlweise entweder einzelnen Objekten oder Gruppen zugesprochen werden. Während im ersten Fall die Anweisung '`SET PROPERTY`' verwendet wird benutzt man im zweiten Fall '`SET GROUP_PROPERTY`' bzw. '`SET SUPERGROUP_PROPERTY`'.

```
SET PROPERTY             -> Objekt
SET GROUP_PROPERTY       -> Gruppe
SET SUPERGROUP_PROPERTY  -> Supergruppe
```

Im Folgenden werden ein paar Anweisungen zu den Gruppeneigenschaften als Ueberblick aufgeführt. Zu den Details der einzelnen Parameter in den Funktionen sehen Sie die entsprechende Funktion für einzelnen Objekte '`SET PROPERTY (O1, ...)`'.

Die *object reference* ‘O1’ kann wahlweise ersetzt werden durch:

```
O1 || LAST_GROUP_NR || LAST_SUPERGROUP
```

Der *qualifier* ‘GROUP_PROPERTY’ kann in Abstimmung mit der Objekt Referenz wahlweise ersetzt werden durch ‘SUPERGROUP_PROPERTY’.

```
SET GROUP_PROPERTY(O1, VELOCITY, {X|Y|Z}, vAbs)
SET GROUP_PROPERTY(O1, C_DAMPING, BENDING, double)
SET GROUP_PROPERTY(O1, C_DAMPING, NORMAL, double)
SET GROUP_PROPERTY(O1, C_DAMPING, TORSION, double)
SET GROUP_PROPERTY(O1, C_LINK, BENDING, double)
SET GROUP_PROPERTY(O1, C_LINK, NORMAL, double)
SET GROUP_PROPERTY(O1, C_LINK, TORSION, double)
SET GROUP_PROPERTY(O1, DENSITY, double)
SET GROUP_PROPERTY(O1, ROTATION_LOCKED, X|Y|Z, bool,
                       ALL || FIRST_LAST)
SET GROUP_PROPERTY(O1, COLOR_STD, 4)
SET GROUP_PROPERTY(O1, COLOR_RGB, 255, 0, 0)
SET GROUP_PROPERTY(O1, C_LINK, STRAIN_LIMIT,
                       PERCENT || ABSOLUTE, double)
SET GROUP_PROPERTY(O1, C_LINK, STRENGTH_CALC, mode, value, fac)
SET GROUP_PROPERTY(O1,C_LINK,STRENGTH_ULTIMATE,mode,value,fac)
SET GROUP_PROPERTY(O1, C_LINK, UNLIMITED, bool)
SET GROUP_PROPERTY(O1, C_LINK, OVERLOAD_ACTION, action)
SET GROUP_PROPERTY(O1, C_LINK, MATERIAL_MODEL, modelNr)
SET GROUP_PROPERTY(O1, C_LINK, CTR120, K_ctr, K_r)
SET GROUP_PROPERTY(O1, C_INTERACT_LIN, double)
SET GROUP_PROPERTY(O1, C_INTERACT_QUAD, double)
SET GROUP_PROPERTY(O1, FORCE_EXT, {X|Y|Z}, double)
SET GROUP_PROPERTY(O1, INTERACT_METHOD, ELASTIC, value)
SET GROUP_PROPERTY(O1, INTERACT_MODE, {ACTIVE,PASSIVE,..})
SET GROUP_PROPERTY(O1, NAME, “groupname”)
SET GROUP_PROPERTY(O1, YIELD_MODEL, modelNr)
SET GROUP_PROPERTY(O1, SIM_MEMBER, FALSE)
```

Ueberblick

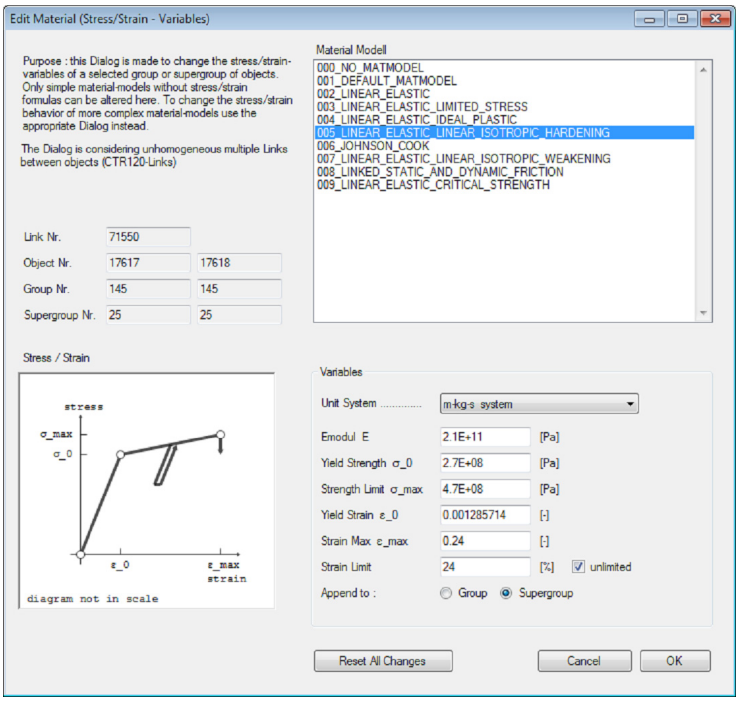


FIGURE 8. Der Dialog 'Edit Material (Stress/Strain-Variables)'

Um die sonar-script-Anweisungen zu den Materialmodellen ins Spiel zu bringen orientieren wir uns am Dialog aus dem sonar-LAB Programm welcher dem gleichen Zweck dient. Der Benutzer hat nach der Selektion eines Objektes einer Gruppe den abgebildeten Dialog vor sich. An dieser Stelle sei nochmals wiederholt, dass der Einsatz dieser Materialmodelle nur in Zusammenhang mit Objektgruppen bestehend aus einer grösseren Anzahl von Objekten einen Sinn macht. Es geht um die kontinuierliche Belastung und Verbiegung von Objektstrukturen.

- Rechts oben wählt der Benutzer ein Modell. Die Anzahl der zur Verfügung stehenden Materialmodelle wird im Laufe der Zeit stetig anwachsen.
- Darunter hat er die Möglichkeit, je nach Modell mehr oder weniger Materialdaten einzugeben. Wir haben zu diesem Zweck beispielhaft das Materialmodell-005 ausgewählt, welches alle Variablen bzw. alle Eingabefelder zur Verfügung stellt.

Wir wollen nun der Reihe nach die entsprechenden sonar-script-Anweisungen erklären.

- Materialmodell
- Elastizitätsmodul
- Streckgrenze
- Bruchspannung

- Dehngrenze (%)
- Aktion bei Ueberlast
- Unbeschränkte Dehnung

Materialmodell

Syntax

```
SET GROUP_PROPERTY(O1, C_LINK, MATERIAL_MODEL, modelNr)
```

O1: Objekt Referenz. O1 ist hier ein Repräsentant einer Objektgruppe
C_LINK, MATERIAL_MODEL: reservierte Worte
modelNr: ganze Zahl (Materialmodell-Nr = 1, 2, 3, ...)

Beschreibung

Für alle Objekt welche zur selben Gruppe gehören wie die Objektreferenz 'O1' soll das Materialmodell mit der Nummer 'modelNr' eingestellt werden. Obwohl wir die Modellnummer im Dialog mit "005_LINEAR_ELASTIC_LINEAR_ISOTRPIC_HARDENING" bezeichneten, geben wir in der obigen sonar-Anweisung für die 'modelNr' einfach die Zahl '5' an.

Beispiel

```
SET GROUP_PROPERTY(O1, C_LINK, MATERIAL_MODEL, 5)
```

Elastizitätsmodul (young modulus)

Syntax

```
SET VALUE(E_MODUL = double, GROUP_NR, LAST_GROUP_NR, f)
SET VALUE(E_MODUL = double, SUPERGROUP_NR, LAST_SUPERGROUP_NR, f)
```

E_MODUL, GROUP_NR, LAST_GROUP_NR, SUPERGROUP_NR, LAST_SUPERGROUP_NR:
reservierte Worte
f: reduction factor [0..1]: mehrere Normal-Links zwischen 2 Objekten

Beschreibung

Der E-Modul nach dem Gesetz von R.Hooke postuliert die Proportionalität zwischen relativer Längenänderung und Normalspannung in einem beschränkten Bereich der Belastung. Alle Link-Konstanten zwischen den Objekten der Gruppe werden so berechnet, dass die Elastizität dem gesetzten Wert entspricht. Dabei werden ggf. sog. Mehrfachlinks mit einem Faktor f berücksichtigt.

Beispiel

```
SET VALUE(E_MODUL = 2.1, GROUP_NR, LAST_GROUP_NR, 0.333)
```

Den Wert 0.333 würden wir setzen, wenn jeweils 3 normale Links die Objekte verbinden. Für einen Wert von 2.1E+11 Pascal, wie er z.B. für Stahl gilt, ergibt sich im [cm-g-us]-System der Wert 2.1

$$2.1E+11\ Pa = 2.1E+11\ N/m^2$$
$$= 2.1E+11\ kg\ m/s^2m^2 = 2.1E+11\ E+3\ g\ E+2\ cm\ /\ (E+12\ \mu s^2\ E+4\ cm^2)$$
$$= 2.1\ E+0\ g/\mu s^2cm$$

Streckgrenze (yield strength)

Syntax

```
SET GROUP_PROPERTY(O1, C_LINK, STRENGTH_CALC, mode, value, fac)
```

O1: Objekt Referenz. O1 ist hier ein Repräsentant einer Objektgruppe
C_LINK. STRENGTH_CALC: reservierte Worte
mode: SPECIFIC || ABSOLUTE,
value: yield strength [cm-g-us]-System

fac: reduction factor [0..1]: mehrere Normal-Links zwischen 2 Objekten

Beschreibung

Gemeint ist streng genommen die Spannung an der Streckgrenze. Es ist der Grenzwert im Spannungs-Dehnungs-Diagramm, wo das Hooke'sche Gesetz seine Gültigkeit verliert und eine bleibende plastische Verformung einsetzt. Der Wert für die Streckgrenze kann auf zwei unterschiedliche Arten festgelegt werden: absolut oder spezifisch (`ABSOLUTE, SPECIFIC`). 'Absolut' bezieht sich auf die gesamte Querschnittsfläche am Ort wo der Link eingebaut wird, während 'spezifisch' sich auf die Einheitsfläche bezieht. Der Reduktionsfaktor 'fac' berücksichtigt Mehrfachlinks an der fraglichen Querschnittsfläche. Würden zwei Zylinder stirnseitg von 3 Links zusammengehalten, dann würde man für 'fac' den Wert '0.333' einsetzen um die Festigkeit gleichmässig auf alle drei Links zu verteilen.

Bruchspannung

Syntax

`SET GROUP_PROPERTY(O1, C_LINK, STRENGTH_ULTIMATE, mode,value,fac)`

O1: Objekt Referenz. O1 ist hier ein Repräsentant einer Objektgruppe
C_LINK. STRENGTH_ULTIMATE: reservierte Worte
mode: SPECIFIC || ABSOLUTE
value: ultimate strength [cm-g-us]-System
fac: reduction factor [0..1]: mehrere Normal-Links zwischen 2 Objekten

Beschreibung

Die Bruchspannung ist die Spannung am Ende der Dehnphase, also an der Dehngrenze wenn das Material bricht und die Spannung auf Null absinkt. Dieser Vorgang ist irreversibel.

Dehngrenze (strain limit)

Syntax

`SET GROUP_PROPERTY(O1, C_LINK, STRAIN_LIMIT, mode, value)`

O1: Objekt Referenz. O1 ist hier ein Repräsentant einer Objektgruppe
C_LINK. STRAIN_LIMIT: reservierte Worte
mode: PERCENT || ABSOLUTE
value: Dehngrenze [cm-g-us]-System

Beschreibung

Die Dehngrenze ist der Ort wo das Versagen des Materials eintritt. In der Regel benutzen wir in dieser Anweisung den Parameter 'PERCENT' und geben die Dehngrenze in Prozent der relativen Dehnung an.

Dehnung: $= \Delta L/L_0$
Dehnung in Prozent: $100 * \Delta L/L_0$

Die absolute Dehngrenze (Parameter `ABSOLUTE`) ist im Gegensatz dazu eine Verlängerung in [cm].

Aktion bei Ueberlast

Syntax

`SET GROUP_PROPERTY(O1, C_LINK, OVERLOAD_ACTION, action)`

O1: Objekt Referenz. O1 ist hier ein Repräsentant einer Objektgruppe
C_LINK. OVERLOAD_ACTION: reservierte Worte
action: {NOACTION || BREAKUP || COLORING || SIGNAL}

Beschreibung

Für den Fall, dass während einer Simulation ein Bruch eintritt, kann der Benutzer die resultierende Aktion als Folge dieser Ueberlastung vorprogrammieren. Nur mit dem Parameter '`BREAKUP`' tritt tatsächlich ein

Bruch ein. Die anderen Aktionen dienen mehr dazu, einen eintretenden Bruch symbolisch zur Kenntnis zu bringen, sei es durch einen Ton oder durch das Einfärben des betreffenden Links mit roter Farbe.

Unbeschränkte Dehnung

Syntax

```
SET GROUP_PROPERTY(O1, C_LINK, UNLIMITED, bool)
```

O1: Objekt Referenz. O1 ist hier ein Repräsentant einer Objektgruppe
C_LINK. UNLIMITED: reservierte Worte
bool: {TRUE || FALSE}

Beschreibung

Falls diese Anweisung mit dem Parameter 'TRUE' auf die referenzierte Objektgruppe angewendet wird, dann wird ein bevorstehender Materialbruch ignoriert. Die letzte Phase des Spannungs-Dehnungs-Diagramms wird ggf. weiter linear fortgesetzt, als ob nichts geschehen wäre. Das System benutzt also in diesem Fall zwar die gesamte Spannungs-Dehnungskurve, so wie sie mit allen Parametern definiert wurde, aber ignoriert einfach den bevorstehenden Bruch. Die exakte Definition des Materialmodells wird also damit nicht überflüssig.

Funktionsparameter

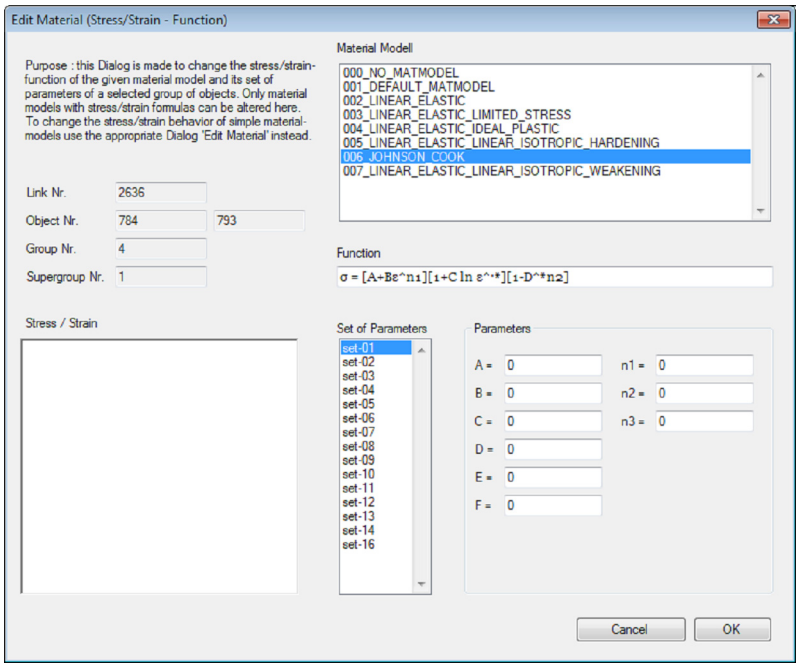


FIGURE 9. Der Dialog 'Edit Material (Stress/Strain-Function)'

Eine andere Gruppe von Materialmodellen lassen sich nicht mit den Standardwerten des letzten Kapitels beschreiben. Sie haben ihre eigenen Funktionen für die Beschreibung der Spannungs-Dehnungs-Abhängigkeit. Ein Beispiel einer solchen Funktion ist die 'Johnson-Cook-Funktion' welche im Dialog angeschrieben ist. Diese Funktion ist für grosse anhaltende Krafteinwirkungen geeignet.
Für jede implementierte Materialfunktion, egal wie sie mathematisch aufgebaut ist, stehen letztlich bis zu 9 Parameter zur Verfügung, mit denen der Benutzer die betreffende Funktion spezifizieren kann, analog wie das oben mit der Johnson-Cook-Funktion angeschrieben wurde. Mit der folgenden Anweisung lässt sich ein sog. 'Set' von Parametern speichern.

Damit hat der Benutzer die Möglichkeit, auch für die gleiche Materialfunktion (z.B. Johnson-Cook) für jedes benutzte Material einen eigenen Satz von Johnson-Cook-Parametern speichern. Wenn der Benutzer später für einen Link eine Materialmodell abruft, dann wählt er nebst der eigentlichen Materialfunktion auch noch das gewünschte Set der Parameter.

Syntax

```
SET VALUE (MATERIAL_PARAM, Index, A, B, C, D, E, F, n1, n2, n3)
```

MATERIAL_PARAM: reserviertes Wort
Index: 'Set'-Nummer
A, B, C, D, E, F, n1, n2, n3: zu setzende Parameter (Fliesskommawerte)

Parameter (A...F, n1...n3) welche Sie für das gewählte Modell nicht benutzen, werden einfach auf Null gesetzt.

Vergleichspannungshypothese (yield model)

```
SET GROUP_PROPERTY(O1, YIELD_MODEL, modelNr)
```

modelNr hat die folgenden Zahlenwerte {1..7}:
1 : Rankine Maximum Principal Stress Theory
2 : St.Venant Maximum Principal Strain Theory
3 : Tresca Maximum Shear Stress Theory
4 : Beltrami Haigh Total Strain Energy Theory
5 : Von Mises Distortion Energy Theory
6 : Mohr Coulomb yield criterion
7 : Drucker Prager yield criterion

Als default-Modell wird der Parameter 1 (Rankine) verwendet. Bei reiner Zuglast liefert dieses Modell gute Näherungswerte.

Kontrollsysteme

Wir unterscheiden in der sonar-Software die nachfolgenden Kontrollsysteme und orientieren uns dabei auch an den entsprechenden Programm-Dialogen die wir zum Vergleich heranziehen. Einige Kontrollsysteme sind nur als Dialog in sonar-SIM verfügbar, mit sonar-script können sie z.T. aber bereits in sonar-LAB erzeugt werden.

Kontrollsysteme

- Punktkurven
  - einfache Punktkurve
  - externe Punktkurve (File)
- Automatisches Kontrollsystem
- Kontrollsystem mit Formeln
- Lineare Funktion der Zeit
- Bedingung für Simulations-Stop
- Beschränkung der Geschwindigkeit eines bestimmten Objektes
- sonar script Kontrollsystem

Punktkurve

Punktkurven nennen wir eine durch Strecken verbundene Folge von Punkten (x,y) in einem kartesischen Koordinatensystem welche eine funktionale Abhängigkeit beschreiben. Sowohl für die X- als auch für die Y-Achse dieser Kurve kann der Benutzer eine Variable aus einem beschränkten Angebot auswählen. Es stehen zwei Methoden zur Verfügung Punktkurven zu definieren. Eine einfache Kurve mit wenigen Datenpunkten kann direkt zusammen mit den zugehörigen Punkten vorgegeben werden, während Kurven mit vielen Datenpunkten in einem externen File vorbereitet und mit der zweiten der beiden nachfolgenden Anweisungen geladen werden. Vergleichen Sie dazu die beiden Dialoge:

- sonar_LAB / Menu / Functions / New Point Curve...
- sonar_LAB / Menu / Functions / New External Point Curve...

Syntax

```
CREATE POINT_CURVE(O1, SIMPLE, varX, varY, wx, wy, wz, mode,
                   activation, nPoints, x1, y1, ... , xn, yn)
```

O1: Objekt Referenz. Bezugsobjekt welches durch die Kurve gesteuert wird.
SIMPLE: reserviertes Wort
varX, varY: Bezeichner der beiden Variablen für die X- und Y-Achse (reservierte Worte)
wx, wy, wz: Vektor = Wirkungsrichtung im Raum
mode: Was zu tun ist wenn das Ende der Punktkurve erreicht wird: 0: stop; 1: repeat periodically
activation: Aktivierung der Kurve = Bool'scher Wert {TRUE, FALSE}
nPoints: Anzahl Punkte in der Punktkurve (integer)
x1, y1, ... , xn, yn: kartesische Koordinaten der Punkte (nPoints an der Zahl)

Beschreibung

Die Anweisung erzeugt eine neue einfache Punktkurve und stellt alle notwendigen Informationen, inklusive der Punkte selbst, in einer einzigen Anweisung zur Verfügung.

Syntax

```
CREATE POINT_CURVE(O1, FILENAME, "filename", varX, varY, wx, wy,
                   wz, mode, activation)
```

O1: Objekt Referenz. Bezugsobjekt welches durch die Kurve gesteuert wird.
FILENAME: reserviertes Wort
"filename": der vollständige Filepfad des Datenfiles mit den Datenpunkten
varX, varY: Bezeichner der beiden Variablen für die X- und Y-Achse (reservierte Worte)
wx, wy, wz: Vektor = Wirkungsrichtung im Raum
mode: Was zu tun ist wenn das Ende der Punktkurve erreicht wird: 0: stop; 1: repeat periodically
activation: Aktivierung der Kurve = Bool'scher Wert {TRUE, FALSE}

Beschreibung

Statt dass die Punktepaare explizit angeschrieben werden, wird der vollständige Filename eines Files angeschrieben, welches die Punkte enthält. In folgendem Beispiel ist die gesamte Anweisung auf einer einzigen Linie zu schreiben.

Beispiel

```
CREATE POINT_CURVE(O3, FILENAME,
"C:\sonar Import\Moment_Welle.txt", TIME, MOMENT_FORCE_EXT,
0, 0, -1, 1, FALSE)
```

Das Beispiel erzeugt eine neue Punktkurve in Form eines Drehmomentes als Funktion der Zeit welches auf Objekt 'O3' wirkt. Die Wirkungsrichtung ist die Z-Achse im Uhrzeigersinn und das Kontrollsystem bleibt vorläufig noch ausgeschaltet. Wenn während der Simulation das Ende der Kurve erreicht werden sollte, dann beginnt die Kurve wieder von vorne.

Automatisches Kontrollsystem

Syntax

```
CREATE CONTROLSYSTEM_AUTO(o1,v1,c1,vl1,vl2, o2,v2,c2,vl3,vl4, a)
```

Angetriebenes System (driven object):
O1: Objekt Referenz. Bezugsobjekt welches durch die Kurve gesteuert wird.
v1: variable (reserviertes Wort)
c1: component (reserviertes Wort)
v11: Nominalwert, Zielwert, Sollwert
v12: aktueller Wert

Referenzobjekt, Treibendes Objekt, Antriebssystem (driving object):
O2: Objekt Referenz. Bezugsobjekt von welchem Werte abgegriffen werden.
v2: variable (reserviertes Wort)
c2: component (reserviertes Wort)
v13: Startwert
v14: aktueller Wert
a: Aktivierung (Boolscher Wert {TRUE, FALSE})

Beschreibung

Ein automatisches Kontrollsystem dieser Form hat eine gewisse Flexibilität in seiner Wirkungsweise, weil es permanent seinen Ist-Zustand mit dem angestrebten Sollzustand vergleicht und seine Wirkung gezielt so einsetzt um den Sollzustand zu erreichen. Das Kontrollsystem schaltet sich folglich automatisch ein- und aus und kann ggf. auch seine Wirkungsrichtung ändern. Im Prinzip besteht das Kontrollsystem aus einem angetriebenen Objekt auf welches z.B. ein Drehmoment einwirkt und einem Referenzobjekt welches darüber entscheidet ob und in welche Richtung das Drehmoment wirken soll indem es seinen eigenen Zustand bzw. eine eigene Variable als Referenz benutzt um diese Entscheidung zu treffen. Die Objektreferenzen 'O1' und 'O2' dürfen auch gleich sein.
Das betreffende Objekt würde dann z.B. im Rahmen des Kontrollsystems sagen, "wenn meine eigene Drehzahl vom angestrebten Sollwert abweicht, dann lass ich ein Drehmoment auf mich wirken". Es könnte aber auch heissen: "Wenn das Drehmoment eines anderen Objektes vom Sollwert abweicht, dann soll ein Drehmoment auf mich wirken".
Sehen Sie dazu auch den Dialog 'Automatic Control Systems' in 'sonar-Sim'.

Zwangsbewegung (constraint movement)

```
CREATE CONTROLSYSTEM_CM(o1, x1,y1,z1,t1, x2,y2,z2,t2, active,
stop, mode)
```

O1: Objekt Referenz. Zwangsbewegtes Objekt.
x1,y1,z1: Startposition des Objektes zur Zeit t1
x2,y2,z2: Endposition des Objektes zur Zeit t2
active: true || false
stop: anhalten bei Erreichen der Endposition (true || false)
mode: parameter {0, 1} 0: no further instructions; 1: ignore all endposition information and take the actual position of object o1 and the actual time for t1.

Beschreibung

Die Funktion überführt das referenzierte Objekt o1 ausgehend von seiner aktuellen oder gegebenen Ausgangslage im Zeitraum [t1...t2] in die Endlage. Bei diesem Vorgang wird die aktuelle Position während der Simulation laufend durch lineare Interpolation berechnet.

$$P_{ik} = P_{1k} + ((T_{ik}-T_{1k})/(T_{2k}-T_{1k}))*(P_{2k}-P_{1k}) \quad \textbf{(EQ 1)}$$

Pi : Position; P1: start; P2: end
T : Zeit; T1: start; T2: end
i : Rechenzyklus
k : Koordinate x, y, z

Gravitationsfeld Erzeugen

Syntax

```
CREATE FIELD (GRAVITATION, nx, ny, nz, b)
```

GRAVITATION: reserviertes Wort
nx, ny, nz: Richtungsvektor des Gravitationsfeldes in kartesischen Koordinaten
b: Gravitationsbeschleunigung

Beschreibung

Die Anweisung erzeugt ein konstantes und isotropes Gravitationsfeld mit einer definierten Wirkungsrichtung. Das Feld ist deshalb für alle Modelle im Labormassstab bis zu Grossanlagen geeignet und ist das Standard-Gravitationsfeld in sonar. Normalerweise wirkt der Gravitationsvektor in den Modellen senkrecht nach unten, man hat aber die Freiheit auch gedrehte Modelle zu definieren indem dieser Vektor in eine andere Richtung zeigt. Wenn Modelle in astronomischen Dimensionen aufgesetzt werden ist dieses Gravitationsfeld nicht mehr geeignet und der Einsatz von zentralen Gravitationsfeldern notwendig. Die folgende Anweisung erzeugt ein Gravitationsfeld im Labormassstab wie es für die meisten Benutzer immer wieder eingesetzt werden wird. Der Richtungsvektor zeigt in minus-y-Richtung, d.h. am Bildschirm senkrecht nach unten und sein Wert entspricht einem mittleren Wert der Erdbeschleunigung, angegeben im [cm-g-µs]-System.

Beispiel

```
CREATE FIELD(GRAVITATION, 0.0, -1.0, 0.0, 9.81E-10)
```

globale Zustände ein/ausschalten

In diesem Abschnitt setzen wir den Zustand von übergeordneten bool'schen Variablen welche sich nicht auf Objekte oder Objektgruppen, sondern auf das ganze Modell als Ganzes beziehen.

Syntax

```
SET STATE (FRICTION, bool)
SET STATE (GRAVITATION, bool)
```

FRICTION, Gravitation, : reservierte Worte
bool: Boolscher Wert. {ON, OFF}. Es kann auch {TRUE, FALSE} verwendet werden.

Beschreibung

Die Anweisung schaltet einen globalen Zustand im Modell ein oder aus. Die Anweisungen sind so gesehen sog. 'Hauptschalter' für den betreffenden Zustand. Wird z.B. die Reibung mit der ersten Anweisung ausgeschaltet, dann bedeutet dies, dass grundsätzlich keine Reibungen mehr berechnet werden, welcher Art auch immer und zwischen welchen Objekten diese auch gesetzt sein mögen. Per default sind die Eigenschaften ausgeschaltet.

Syntax

```
DESELECT ALL
```

Beschreibung

Alle Selektionen, ungeachtet der Objekttypen und der Zusammensetzung der Elemente, werden deaktiviert. Es gibt anschliessend nichts mehr was selektiert wäre. Diese Anweisung wird in 'sonar script' sehr häufig eingesetzt

um zwischen den einzelnen Handlungen sicherzustelle, dass bei neuen Selektionen die alten gelöscht sind. Die Anweisung ‘DESELECT ALL’ entspricht im Prinzip einem Mausklick ins Leere in einer Ansicht des Modells.

globale Werte setzen

Syntax

```
SET VALUE (C_FRICTION_DYNAMIC = double)
SET VALUE (C_FRICTION_STATIC = double)
SET VALUE (TIMESTEP_MAX = double)
```

reservierte Worte :
C_FRICTION_DYNAMIC: Gleitreibungskoeffizient
C_FRICTION_STATIC: Haftreibungskoeffizient
TIMESTEP_MAX: maximal zulässiger Zeitschritt während der Simulation

Beschreibung

Mit diesen Anweisungen werden allgemeine modellspezifische Variablen gesetzt. Den betreffenden globalen Variablen werden durch die Anweisungen bestimmte neue Zahlenwerte zugewiesen, welche solange gültig bleiben, bis ggf. etwas anderes definiert wird. Die Reibungswerte betreffen nur die allgemeinen globalen Werte, d.h. für Objekte für die keine weitergehenden uni- oder bilateralen Reibwerte festgelegt wurden. Die Variable ‘TIMESTEP_MAX’ setzt für den Zeitschritt eine obere Schranke. Bei einer autom. Steuerung des Zeitschrittes kann dieser aber auch kleiner bleiben, falls die Automatik dies für notwendig erachtet.

Seil (Blockmodell)

Syntax

```
CREATE CABLE BLOCKMODEL (x1,y1,z1, x2,y2,z2,
                         n0, w0, ro, CTR120, Kc, Kr, C1, C2)

CREATE CABLE BLOCKMODEL (x1,y1,z1, x2,y2,z2,
                         n0, w0, ro, K1, K2, K3, C1, C2)
```

x1, y1, z1, x2, y2, z2: kartesische Koordinaten der Endpunkte des Seils
n0: Anzahl Element längs dem Seil
w0: Drahtdurchmesser
ro: durchschnittliche Materialdichte des Seils auf den Aussendurchmesser bezogen
K1, K2, K3: Link-Konstanten bezüglich Streckung, Biegung, Torsion (outdated)
Kc: center Link, Kr: radial Links (default)
C1, C2: lineare und quadratische Interaktions-Konstanten

Beschreibung

Ein Blockmodell eines Seils ist eine Vereinfachung. Das Seil besteht in diesem Modell nicht aus inneren Litzen und Drähten aus denen sich das Seil aufbaut und die ihrerseits modelliert werden, sondern man vereinfacht den gesamten Seilaufbau zu einer durchschnittlichen homogenen Masse mit dem gegebenen Aussendurchmesser des Seils. Das Seil besteht in diesem Modell aus einer Kette von Zylindern, welche untereinander elastisch verbunden sind. Man achtet aber darauf, dass das Seil als Ganzes trotzdem den wirklichen physikalischen Eigenschaften entspricht. Zu diesem Zweck wird die Wahl der verschiedenen Steifigkeiten dem Benutzer überlassen, welcher diese explizit setzen kann (K1, K2, K3) resp. (Kc, Kr). Näherungsweise setzt der Benutzer diese Parameter gemäss den folgenden Ueberlegungen:

Normalerweise ist K1 resp. Kc die wichtigste Grösse und man setzt diese longitudinale Steifigkeit so, dass sie der Summe der Steifigkeiten der einzelnen Drähte entspricht. Die Biegesteifigkeit ist bereits schwieriger abzuschätzen. Ein Wert von 1.5 mal der Summe der Biegesteifigkeiten der Einzeldrähte könnte eine grobe Näherung sein. Der Faktor von 1.5 bringt auf diesem Weg die Reibung unter den Drähten ins Spiel. Wenn man es genauer wissen will, ist wahrscheinlich ein Versuch angebracht, wo man ein Stück eines Musterseils über eine Tischkante ragen lässt und den Durchhang misst. Eventuell muss man das Seil am Ende noch zusätzlich belasten um einen messbaren Wert zu bekommen. Anschliessend kann man das Ganze in einem Testmodell in sonar identisch simulieren. In der Simulation ändert man schliesslich die Biegesteifigkeit des Seilmodells solange bis Uebereinstimmung vorliegt.

Die Steifigkeit für die Streckung, Biegung und Torsion eines einzelnen Drahtes berechnet sich wie folgt:

$$\text{Streckung} : K1 = E * Ae / eL \qquad \textbf{(EQ 2)}$$

$$\text{Biegung} : K2 = \pi * d^4 * E / (64 * eL) \qquad \textbf{(EQ 3)}$$

$$\text{Torsion} : K3 = \pi * d^4 * G * \ / (32 * eL) \qquad \textbf{(EQ 4)}$$

E: Emodul
G: Schub- bzw. Gleitmodul

d: Drahtdurchmesser
Ae: Drahtquerschnittsfläche = $d^2 * \pi / 4$
eL: Elementlänge = Länge der einzelnen Zylinder aus denen sich der Draht aufbaut

Man muss bei diesen Rechnungen darauf achten, dass sie konsequent im sog. [cm-g-µs]-System durchgeführt und die Resultatwerte in dieser Form in die Anweisung zur Erzeugung des Blockmodells eingesetzt werden.

Seile dieser Art eignen sich gut für die Simulation von ganzen Anlagen welche Seile benutzen wie Seil- und Gondelbahnen, Aufzug- und Förderanlagen, Schutzanlagen, usw. Ueberall dort wo der innere Aufbau nicht im Zentrum der Betrachtungen steht, sondern das makroskopische Verhalten von Interesse ist, sind diese Seilmodelle gut geeignet für Simulationen.

Biegsamer Draht, Drahtfeder

Syntax

```
CREATE WIRE_LINE (x1,y1,z1, x2,y2,z2,
                  n0, w0, ro, E, G, C1, C2, mode)
```

x1, y1, z1, x2, y2, z2: kartesische Koordinaten der Endpunkte des Seils
n0: Anzahl Elemente längs dem Draht
w0: Drahtdurchmesser
ro: Materialdichte
E, G: Emodul, Gleit- bzw. Schubmodul
C1, C2: lineare und quadratische Interaktions-Konstanten
mode: CTR120 (default), EXCL120, (NBT outdated).

Beschreibung

Aehnlich wie mit dem Blockmodell für ein Seil wird hier 'de facto' eine Kette von Zylindern modelliert welche untereinander elastisch verbunden sind. In dieser Anweisung gibt der Benutzer allerdings keine Steifigkeiten für einen bestimmten Link an, sondern hat die Wahl welchen Link-Typ er verwenden will. Er bestimmt die Steifigkeit des Drahtes durch die Angabe des Emoduls und des Gleitmoduls. Unter der Voraussetzung, dass man mit dieser Anweisung die gleichen Parameter setzt wie in den Anwesisungen zur Erzeugung eines Blockmodells weiter oben, endet das erzeugte Modell beim gleichen Resultat wie wenn man das Blockmodell für Seile verwendet hätte. Auch mit dieser Anweisung wird default-mässig die sog. CTR120 Link-Kombination verwenden, welche für Drähte am besten geeignet ist. Drähte mit CTR120-Links sind numerisch stabiler als NBT-Links. Mit CTR-120 Links kann ein Draht in einer Simulation auch plastisch deformiert werden. Und zwar nicht nur longitudinal sondern auch was die Biegung betrifft.

Biegsamer Drahtring

```
CREATE WIRE_RING ( R, w0, n0, ro, E, G, C1, C2, mode)
```

R: mittlerer Durchmesser des Drahtringes
w0: Drahtdurchmesser
n0: Anzahl Elemente längs dem Ring
ro: Materialdichte
E, G: Emodul, Gleit- bzw. Schubmodul
C1, C2: lineare und quadratische Interaktions-Konstanten
mode: CTR120 (default), EXCL120, (NBT outdated).

Alles was für den biegsamen Draht gesagt wurde gilt auch für den Drahtring. Für den Parameter n0 ist 32 ein guter Wert. Weniger als 12 sollte für die Anzahl Objekte nicht eingesetzt werden. Will man Drahtringe aneinander hängen, dann müssen diese nach der Erzeugung gegenseitig schräg gestellt werden, damit ihre Volumina sich nicht durchdringen sondern äusserlich bestenfalls berühren.

Blattfeder

Syntax

```
CREATE SPRING (LEAF, STANDARD, dL, dw, dh,
               n0, ro, E, G, C1, C2, mode)
```

LEAF, STANDARD: reservierte Worte
dL, dw, dh: Länge, Breite und Dicke der Blattfeder
n0: Anzahl Quader-Elemente längs der Blattfeder
ro: Materialdichte
E, G: Emodul, Gleit- bzw. Schubmodul
C1, C2: lineare und quadratische Interaktions-Konstanten
mode: NNB, NNNN

Beschreibung

Es wird eine sog. Standard-Blattfeder mit einem rechteckigen Querschnitt erzeugt. Die Feder wird längs der Z-Achse bereitgestellt mit einem der beiden Endpunkte (Querschnittsmittelpunkt) im Ursprung. Die Feder kann anschliessend mit den üblichen Bewegungsfunktionen für Gruppen in ihre endgültige Lage gedreht und verschoben werden. Die Feder befindet sich beim Erzeugen in einem ungebogenen Zustand. Wird als Ausgangspunkt in einem Modell eine gebogene Feder benötigt, dann muss diese separat mit einer Hilfssimulation (externe Kraft) in die gewünschte Lage gebracht werden. Am besten würde man diesen Vorgang mit der betreffenden Feder allein in einem eigenen Modell durchführen und die einbaufertige Feder schliesslich mittels 'Merge' in das Zielmodell einsetzen.

physikal. Zug- oder Druckfeder

Syntax Zugfeder

```
CREATE SPRING (HELICAL, TENSION, STANDARD,
               D1, L0, Ln, d, ro, wd, C1, C2, F0, Fn)
```

Syntax Druckfeder

```
CREATE SPRING (HELICAL, COMPRESSION, STANDARD,
               D1, L0, Ln, d, ro, wd, C1, C2, F0, Fn)
```

HELICAL, TENSION, COMPRESSION, STANDARD: reservierte Worte
D1: Federdurchmesser (gemessen über die Drahtmitte)
L0, Ln: die Nulllänge und die gespannte Länge gemäss dem Federdatenblatt
d: Drahtdurchmesser
ro: Materialdichte
wd: Anzahl wirksame Windungen
C1, C2: lineare und quadratische Interaktionskonstante
F0, Fn: Die Kräfte bei L0 und Ln gemäss dem Federdatenblatt

Beschreibung

Die Anweisungen konstruieren komplette physikalische Schraubenfedern mit Standard-Oesen an den Enden, welche sich dazu eignen, die Federn mittels Bolzen zu halten. Der Aufbau der Federn geschieht mit Torussegmenten und Zylindern. Die erzeugten Federn bekommen automatisch eine eigene Gruppennummer. Mit dieser können allenfalls gewisse Eigenschaften der Feder verändert werden. So bekommt die Feder 'per default' die Interaktionsart 'PASSIVE', was soviel bedeutet wie: Die Feder kann mit anderen Objekten (z.B. mit dem Aufhängebolzen) interagieren, die Drahtwindungen kollidieren aber nicht untereinander. Dies entspricht den meisten Bedürfnissen. Falls man das ändern möchte, dann setzt man anschliessend einfach die betreffende Gruppe auf 'ACTIVE'.

Biegsames Rohr

Syntax

```
CREATE TUBE_LINE (x1,y1,z1, x2,y2,z2,
                  n0, da, di, ro, E, G, C1, C2, mode)
```

x1, y1, z1, x2, y2, z2: kartesische Koordinaten der Endpunkte des Rohrs
n0: Anzahl Elemente längs dem Rohr
da, di: Rohr-Aussen- und Innendurchmesser
ro: Materialdichte
E, G: Emodul, Gleit- bzw. Schubmodul
C1, C2: lineare und quadratische Interaktions-Konstanten
mode: CTR120, EXCL120, (NBT outdated)

Beschreibung

Das Vorgehen ist völlig analog wie bei der Bildung eines Drahtes. Der einzige Unterschied liegt darin, dass hier Zylinder mit Bohrungen (Rohrelemente) verwendet werden statt Vollzylinder. Daraus folgen auch gewisse Einschränkungen welchen dieses Rohrmodell anschliessend unterworfen sein wird. Weil das biegsame Rohr aus starren Rohrelementen besteht, kann dieses zwar verbogen werden, aber das Rohr kann nicht kollabieren. Das Rohr wird auch bei Ueberlast seine runde Form beibehalten und sich allenfalls zwischen den Rohrelementen plastisch verformen oder auseinanderbrechen. Falls man ein kollabierendes Rohr modellieren möchte, dann muss dieses mit sog. Rohrsegmenten (Tube Segments) aufgebaut werden. Rohre neigen zum Kollabieren wenn die Wandungsstärke im Verhältnis zum Rohrdurchmesser sehr gering ist.

Schäkel (shackle)

```
CREATE SHACKLE (SCREW_PIN_ANCHOR, nominal_size)
```

SCREW_PIN_ANCHOR: reserviertes Wort
nominal_size: eine existierende Standardgrösse als Dezimalwert (z.B. 7/8” = 0.875)
Folgende Grössen stehen zur Verfügung: {0.25, 0.3125, 0.375, 0.4375, 0.5, 0.625, 0.75, 0.875, 1.0, 1.125, 1.25, 1.375, 1.5, 1.75, 2.0, 2.5}

Alle Teile des erzeugten Schäkels tragen eine gemeinsame Gruppennummer. Der Schäkel befindet sich vorerst im Ursprung.

Uebersicht ‘Macro Language’ in alphabetischer Reihenfolge

Bemerkung

einige der folgenden Anweisungen (Italic) sind nur in speziellen Modulen zu sonar_LAB verfügbar.

```
ATTACH OBJECT (SELECTION, TRUE || FALSE)
ATTACH OBJECTGROUP (oNr, LAST_GROUP_NR, x, y, z, x2, y2, z2, kc)
ATTACH OBJGRP_OBJGRP (oNr1, oNr2, mode)
BEGIN SCRIPT scriptname || BEGIN MACRO scriptname
CLEAR ALL
CLEAR ELEMENT (E1 || SELECTION)
CLEAR OBJECT (O1 || SELECTION)
CONCATENATE ELEMENTS (E1, RING, x0, y0, n)
CREATE BRAID (
   x1, y1, z1, x2, y2, z2,
   SL, D, nW, nE, d,
   ro, E, G, C1, C2, colorIdx, iActMethodIdx, iActModeIdx,
   matIndex)
CREATE BRAID_LITZE (
   x1, y1, z1, x2, y2, z2,
   SL, D, nL, SL0, SR0,
   d0,n0,e0,
   d1,n1,e1,
   d2,n2,e2,
   d3,n3,e3,
   d, ro, E, G, C1, C2, colorIdx, iActMethodIdx, iActModeIdx,
   matIndex)
CREATE CABLE (
   x1, y1, z1, x2, y2, z2,
   SL_L0, SR_L0, n0_L0, e0_L0, d0_L0,
                 n1_L0, e1_L0, d1_L0,
                 n2_L0, e2_L0, d2_L0,
                 n3_L0, e3_L0, d3_L0,
   SL_L1, SR_L1, n0_L1, e0_L1, d0_L1,
                 n1_L1, e1_L1, d1_L1,
                 n2_L1, e2_L1, d2_L1,
                 n3_L1, e3_L1, d3_L1,
   SL_L2, SR_L2, n0_L2, e0_L2, d0_L2,
                 n1_L2, e1_L2, d1_L2,
                 n2_L2, e2_L2, d2_L2,
                 n3_L2, e3_L2, d3_L2,
   SL_S1, SR_S1, nS1, H_diam_S1,
   SL_S2, SR_S2, nS2, H_diam_S2,
   ro, E, G, C1, C2, colorIdx, iActMethodIdx, iActModeIdx,
   matIndex)
CREATE CABLE_BLOCKMODEL (
   x1,y1,z1, x2,y2,z2,
   n0, wd0,
   ro, K1, K2, K3, C1, C2)
CREATE CHAIN (
   typeNr,e1,nK,nA,o1,o2,o3,o4,nB,o5,o6,o7,o8,b1,b2,S,C1,C2)
```

```
CREATE CLUSTER (PARTICLE_SPHERE, DENSE || CUBIC,
   x0, y0, z0, nx, ny, nz, R1, R2)
CREATE CONTOUR_LINE_ARC(C1, E1 || SELECTION)
CREATE CONTROLSYSTEM_AUTO(o1,v1,c1,vl1,vl2, o2,v2,c2,vl3,vl4,dr)
CREATE ELEMENT (E1, ARC, x0,y0,z0, x1,y1,z1, x2,y2,z2, orient)
CREATE ELEMENT (E1, CIRCLE, cx, cy, cz, nx, ny, nz, R)
CREATE ELEMENT (E1, LINE, x1, y1, z1, x2, y2, z2)
CREATE ELEMENT (E1, POINT, x0, y0, z0)
CREATE ELEMENT (E1, POLYGON,  n); DATA(x1,y1,z1, ..., xn,yn,zn)
CREATE ELEMENT (E1, POLYLINE,  n); DATA(x1,y1,z1, ..., xn,yn,zn)
CREATE ELEMENT (E1, QUAD_STRIP, n); DATA(x1,y1,z1, ..., xn,yn,zn)
CREATE FIBRE   (x1, y1, z1, x2, y2, z2, n0, d0, ro, E, G, C1, C2)
CREATE FIELD(Gravitation, nx, ny, nz, b)
CREATE FIXPOINT (F1, x0, y0, z0)
CREATE FIXPOINT (F1, SELECTION)
CREATE IACT_RULE(O1, O2, bool)
CREATE IACT_RULE(O1, O2, bool, mode1, mode2)
CREATE IACT_RULE(SELECTION, bool, mode1, mode2)
CREATE LINK (K1, NORMAL, O1, O2, E1)
CREATE LINK (K1, NORMAL || BENDING || TORSION, O1, O2, x0,y0,z0)
CREATE LINK (K1, NORMAL, O1, F1)
CREATE LINK (K1, NORMAL || BENDING || TORSION, SELECTION)
CREATE LINK (K1, NORMAL, AUTOMATIC, E1)
CREATE LINK (K1, NORMAL, AUTOMATIC, E1, O1)
CREATE LINK (K1, NORMAL, AUTOMATIC, CTR120, E1)
CREATE LINK (K1, NORMAL, CTR120, O1, O2)
CREATE LINK (K1, U_BREMSE, O1, O2, x1,y1,z1, x2,y2,z2)
CREATE OBJECT (O1, CONE, x0, y0, z0, wx, wy, wz, R, r, dz)
CREATE OBJECT (O1, CONE, x1, y1, z1, x2, y2, z2, r1, r2)
CREATE OBJECT (O1, CUBOID, x0, y0, z0, wx, wy, wz, dx, dy, dz)
CREATE OBJECT (O1, CUBOID, x1, y1, z1, x2, y2, z2)
CREATE OBJECT (O1, CYLINDER, x0, y0, z0, wx, wy, wz, R, dz)
CREATE OBJECT (O1, CYLINDER, x1, y1, z1, x2, y2, z2, R)
CREATE OBJECT (O1, PARTICLE_SPHERE, x0, y0, z0, R)
CREATE OBJECT (O1, PLANE, nx0, ny0, nz0, nx1, ny1, nz1)
CREATE OBJECT (O1, PRISM, E1, EXTRUSION, dz) -- convex only
CREATE OBJECT (O1, PRISM_LINE_ARC, E1 || SELECTION, EXTRUSION,dz)
CREATE OBJECT (O1, PRISM_QUAD_STRIP, E1, EXTRUSION, dz)
CREATE OBJECT (O1, SPHERE, x0, y0, z0, R)
CREATE OBJECT (O1, TORUS, R, r) -- in nullpos.
CREATE OBJECT (O1, TORUS_SEGMENT, R, r, phi) -- phi:[°]
CREATE OBJECT (O1, TUBE, x0, y0, z0, wx, wy, wz, Ra, Ri, dz)
CREATE OBJECT (O1, TUBE_SEGMENT, x1,y1,z1,x2,y2,z2,Ra,Ri,phi,ws)
CREATE OBJECT (O1, TUBE_SURFACE, x0, y0, z0, wx, wy, wz, R, dz)
CREATE POINT_CURVE (
   O1, SIMPLE, varX, varY, wx, wy, wz, mode, activation, nPoints,
   x1, y1, ... , xn, yn)
CREATE POINT_CURVE (O1, FILENAME, "filename", varX, varY, wx, wy,
   wz, mode, activation)
CREATE PROFILE (TYPE_STRIPES, nStripes, nSections,r o, E, G,
   C1, C2, colIdx, iActMethodIdx, iActModeIdx, matIndex,
   x11, y11, z11, x12, y12, z12,
   ...
   x91, y91, z91, x92, y92, z92,
   Lz1, n1, ..., Lz9, n9,
   nLinks,
   k1x, k1y, ..., k18x, k18y)
CREATE PROFILE (TYPE_SHEET_METAL, d, L, nE, nL, ro, E)
CREATE PROFILE (TYPE_UNIFORM_PLATE, x1, x1, z1, x2, y2, z2,
```

```
   nx, ny, nz, ro, E, G, C1, C2, colIdx,
   iActMethodIdx, iActModeIdx, matIndex)
CREATE SPRING (HELICAL, TENSION, STANDARD, D1, L0, Ln, d, ro, wd,
   C1, C2, F0, Fn)
CREATE SPRING (LEAF, STANDARD, dL, dw, dh, n0, ro, E, G, C1, C2,
   mode)
CREATE STRAND (x1, y1, z1, x2, y2, z2,
   n0, d0, e0, n1, d1, e1, n2, d2, e2,
   SL, SR,
   ro, E, G, C1, C2)
CREATE STRAND_COMPLEX (x1, y1, z1, x2, y2, z2,
   SL, SR, n0, e0_SL, wd0,
   n1, e1_SL, wd1, D1, phi1,
   n2, e2_SL, wd2, D2, phi2,
   n3, e3_SL, wd3, D3, phi3,
   n4, e4_SL, wd4, D4, phi4,
   n5, e5_SL, wd5, D5, phi5,
   n6, e6_SL, wd6, D6, phi6,
   ro, E, G, C1, C2, colorIdx, iActMethodIdx, iActModeIdx,
   matIndex)
CREATE TUBE_LINE (x1, y1, z1, x2, y2, z2,
   n0, da, di, ro, E, G, C1, C2, mode)
CREATE WIRE_HELIX (x1, y1, z1, x2, y2, z2,
   SL, SR, e0_SL, wd0, D, phi,
   ro, E, G, C1, C2, DEFAULT || EXCL120)
CREATE WIRE_LINE(x1, y1, z1, x2, y2, z2,
   n0, wd0, ro, E, G, C1, C2, mode)
CREATE WIRE_RING (R, r, n, ro, E, G, C1, C2,
   DEFAULT || EXCL120 || CTR120)
DATA (xi, yi, zi, xj, yj, zj, ... , xn, yn, zn)
DEFORM GRID (G1, ALIGNED, Y, double, double)
DESELECT ALL
DO IF (boolean condition)
DUPLICATE OBJECT (O1 || SELECTION || ALL, INPLACE || bool)
ENHANCE RESOLUTION (x1,y1,z1, x2,y2,z2, eL, E, G) ..,eMode,aMode)
EXPORT GRID (SELECTION, FILENAME, "filename")
FIND OBJECT (O1, x0, y0, z0)
FRAGMENT OBJECT (O1, CUBOID, dx, dy, dz, minSize)
GROUP ELEMENTS (SELECTION)
GROUP ELEMENTS (ALL)
IMPORT COLLECTION_LINE_ARC (E1, FILENAME, "filename")
IMPORT CONTOUR_LINE_ARC (E1, FILENAME, "filename")
IMPORT GRID (O1, FILENAME, "filename", typeNr)
IMPORT POLYLINE (E1, FILENAME, "filename")
JOIN OBJECTGROUP (O1, Emodul)
JOIN OBJECTSUPERGROUP (O1, Emodul)
LOOP FOR (K, i1, i2, step)
MOVE ELEMENT (E1 || SELECTION, MOVE_MATRIX, O1 || SELECTION)
MOVE OBJECT (O1 || SELECTION, ABSOLUTE || RELATIVE, x0, y0, z0,
   wx, wy, wz)
RESET ANGLE (ALL)
RESET F_EXT (ALL)
REVOLVE ELEMENT (G1, SELECTION, GRID_ROT_CONTOUR, w1, w2, nNodes)
REVOLVE SECTION (O1, SELECTION)
ROTATE GRID (G1, cx, cy, cz, wx, wy, wz)
ROTATE OBJECT (O1, cx, cy, cz, wx, wy, wz)
ROTATE OBJECTGROUP(O1, cx, cy, cz, wx, wy, wz)
ROTATE OBJECTSUPERGROUP(O1, cx, cy, cz, wx, wy, wz)
SELECT CONTOUR (C1)
```

```
SELECT ELEMENT (E1)
SELECT FIXPOINT (F1)
SELECT LINK (K1)
SELECT LINK (x0, y0, z0)
SELECT OBJECT (ALL)
SELECT OBJECT (LAST_OBJECT - n)
SELECT OBJECT (O1) || SELECT MICROBE(O1)
SELECT OBJECT (POINT, x1, y1, z1)
SELECT OBJECT (POINT, ALL, x1, y1, z1)
SELECT OBJECT (RECT, XY, x1, x2, y1, y2)
SELECT OBJECT (SPACE, x1, y1, z1, x2, y2, z2)
SELECT OBJECT (COLOR_STD, index)
SELECT OBJECT (COLOR_RGB, red, green, blue)
SET GROUP_PROPERTY(O1, VELOCITY, {X|Y|Z}, vAbs)
SET GROUP_PROPERTY(O1, SUPERGROUP_NR, LAST_SUPERGROUP_NR)
SET GROUP_PROPERTY(O1, C_DAMPING, BENDING, double)
SET GROUP_PROPERTY(O1, C_DAMPING, NORMAL, double)
SET GROUP_PROPERTY(O1, C_DAMPING, TORSION, double)
SET GROUP_PROPERTY(O1, C_LINK, BENDING, double)
SET GROUP_PROPERTY(O1, C_LINK, NORMAL, double)
SET GROUP_PROPERTY(O1, C_LINK, TORSION, double)
SET GROUP_PROPERTY(O1, DENSITY, double)
SET GROUP_PROPERTY(O1, GLUE, c, RMax, TRUE)
SET GROUP_PROPERTY(O1, VISCOSITY, C, N, T, cA, cD, dMax, TRUE)
SET GROUP_PROPERTY(O1, ROTATION_LOCKED, X|Y|Z, TRUE || FALSE,
   ALL || FIRST_LAST)
SET GROUP_PROPERTY(O1, COLOR_STD, 4)
SET GROUP_PROPERTY(O1, COLOR_RGB, 255, 0, 0)
SET GROUP_PROPERTY(O1, C_LINK, STRAIN_LIMIT,
   PERCENT || ABSOLUTE, double
SET GROUP_PROPERTY(O1, C_LINK, STRENGTH_CALC, mode, value, fac)
SET GROUP_PROPERTY(O1, C_LINK, STRENGTH_ULTIMATE, mode,value,fac)
SET GROUP_PROPERTY(O1, C_LINK, UNLIMITED, bool)
SET GROUP_PROPERTY(O1, C_LINK, OVERLOAD_ACTION, action)
SET GROUP_PROPERTY(O1, C_LINK, MATERIAL_MODEL, modelNr)
SET GROUP_PROPERTY(O1, C_LINK, CTR120, K_ctr, K_r)
SET GROUP_PROPERTY(O1, C_INTERACT_LIN, double)
SET GROUP_PROPERTY(O1, C_INTERACT_QUAD, double)
SET GROUP_PROPERTY(O1, FORCE_EXT, {X|Y|Z}, double)
SET GROUP_PROPERTY(O1, INTERACT_MODE, {ACTIVE,PASSIVE,..})
SET GROUP_PROPERTY(O1, SIM_MEMBER, FALSE)
SET GROUP_PROPERTY(O1, YIELD_MODEL, modelNr)
SET POINT (G1, ix, iy, X || Y || Z, double)
SET POINT (G1, ix, iy, x, y, z)
SET PREFERENCE (IACT_CYCLE_CALC_PERIOD, int)
SET PREFERENCE (IACT_DEPTH_LIMIT_ACTIVE, TRUE)
SET PREFERENCE (IACT_DEPTH_LIMIT_MSG, TRUE)
SET PREFERENCE (IACT_DEPTH_LIMIT_VALUE, double)
SET PREFERENCE (IACT_OBJSIZE_SECURITY_FAC, double)
SET PROPERTY(E1, NORMALVECTOR, x1, y1, z1)
SET PROPERTY(E1, NORMALDIRECTION, 1)
SET PROPERTY(E1, INUSE, bool)
SET PROPERTY(K1, ANGLE, {X|Y|Z}, double)
SET PROPERTY(K1, C_DAMPING, double)
SET PROPERTY(K1, C_LINK, double)
SET PROPERTY(K1, MOMENT_FRICTION, double)
SET PROPERTY(O1, ANGULAR_VELOCITY, {X|Y|Z}, double)
SET PROPERTY(O1, BEVEL, ROUND || FACET, double)
SET PROPERTY(O1, C_INTERACT_LIN, 0.0001)
```

```
SET PROPERTY(O1, C_INTERACT_QUAD, 0.0001)
SET PROPERTY(O1, COLOR_STD, 4)
SET PROPERTY(O1, COLOR_RGB, 255, 0, 0)
SET PROPERTY(O1, INTERACT_CONTROLPOINT, x1, y1, z1, TRUE)
SET PROPERTY(O1, DENSITY, 7.8)
SET PROPERTY(O1, END_OF_WIRE, TRUE);
SET PROPERTY(O1, FORCE_EXT, {X|Y|Z}, double)
SET PROPERTY(O1, FRICTION_UNILATERAL, double)
SET PROPERTY(O1, GLUE, TRUE)
SET PROPERTY(O1, GROUP_NR, LAST_GROUP_NR)
SET PROPERTY(O1, INTERACT_DIRECTION, RADIAL_ONLY || AXIAL_ONLY ||
   ALL_DIRECTIONS)
SET PROPERTY(O1, INTERACT_METHOD, ELASTIC, 10)
SET PROPERTY(O1, INTERACT_MODE, {ACTIVE,PASSIVE,NO_INTERACTION})
SET PROPERTY(O1, LONG_CONTOUR_OBJECT, TRUE)
SET PROPERTY(O1, MASS, double)
SET PROPERTY(O1, MOMENT_FORCE_EXT, nx, ny, nz, F)
SET PROPERTY(O1, MOMENT_INERTIA, Ix, Iy, Iz)
SET PROPERTY(O1, MOMENT_INERTIA, FACTOR, double)
SET PROPERTY(O1, NAME, "objectname")
SET PROPERTY(O1, ROTATION_LOCKED, X|Y|Z, TRUE|FALSE)
SET PROPERTY(O1, SIM_MEMBER, FALSE)
SET PROPERTY(O1, SUPERGROUP_NR, LAST_SUPERGROUP_NR)
SET PROPERTY(O1, TRANSPARENCY, integer)
SET PROPERTY(O1, VELOCITY, {X|Y|Z}, double)
SET PROPERTY(O1, VISCOSITY, TRUE)
SET PROPERTY(O1, VISIBILITY, FALSE)
SET PROPERTY(O1, WIREFRAME, TRUE)
SET STATE (BENDING_STRENGTH, ON || OFF, SUPERGROUP_NR,
   LAST_SUPERGROUP_NR)
SET STATE (FRICTION, ON || OFF)
SET STATE (GLUE, ON || OFF)
SET STATE (VISCOSITY, ON || OFF)
SET VALUE (CABLE_CORRECTIONS, HELIX, sNr, wNr, double)
SET VALUE (CABLE_CORRECTIONS, ANGLE, sNr, TRUE|FALSE)
SET VALUE (C_FRICTION_DYNAMIC = double)
SET VALUE (C_FRICTION_STATIC = double)
SET VALUE (C_GLUE = double)
SET VALUE (C_LINK = double, linktype, OBJECTNAME, "objectname")
SET VALUE (E_MODUL= double, SUPERGROUP_NR, LAST_SUPERGROUP_NR, f)
SET VALUE (BENDING_STRENGTH = double, SUPERGROUP_NR,
   LAST_SUPERGROUP_NR, f)
SET VALUE (C_VISCOSITY = double)
SET VALUE (MATERIAL_PARAM, Index, A, B, C, D, E, F, n1, n2, n3)
SET VALUE (MODEL_SCALE, SCREEN_WORLD = value, REAL_WORLD = value
SET VALUE (MODEL_SIZE_FAC_OPENGL = double)
SET VALUE (NEW_GROUP_NR)
SET VALUE (NEW_SUPERGROUP_NR)
SET VALUE (O1 = SELECTION)
SET VALUE (TIMESTEP_MAX = double)
SET VALUE (VISCOSITY_RANGE = double)
SWEEP CROSSSECTION (O1, SELECTION, POLYLINE, CIRCLE, R)
SWEEP CROSSSECTION (O1, SELECTION, LINE_ARC, CIRCLE, R)
TRANSFORM ELEMENTS (E1, CONTOUR)
TRANSFORM ELEMENTS (E1, LINE_SEGMENTS, dL)
TRANSLATE GRID (G1 || SELECTION, ABSOLUTE || RELATIVE, dx,dy,dz)
TRANSLATE OBJECT(O1 || SELECTION, ABSOLUTE || RELATIVE, dx,dy,dz)
TRANSLATE OBJECTGROUP(O1, ABSOLUTE || RELATIVE, dx, dy, dz)
TRANSLATE OBJECTSUPERGROUP(O1, ABSOLUTE || RELATIVE, dx, dy, dz)
```

```
TWIST OBJECT (O1 || SELECTION, Tx, Ty, Tz, dz)
UNGROUP ELEMENTS (ALL)
```

Uebersicht Macro Language nach Funktionsgruppen

Rohdaten

Rohdaten Import

```
IMPORT COLLECTION_LINE_ARC (E1, FILENAME, "filename")
IMPORT CONTOUR_LINE_ARC (E1, FILENAME, "filename")
IMPORT GRID (O1, FILENAME, "filename", typeNr)
IMPORT POLYLINE (E1, FILENAME, "filename")
```

Rohdaten Erzeugen

```
CREATE ELEMENT (E1, ARC, x0,y0,z0, x1,y1,z1, x2,y2,z2, orient)
CREATE ELEMENT (E1, CIRCLE, cx, cy, cz, nx, ny, nz, R)
CREATE ELEMENT (E1, LINE, x1, y1, z1, x2, y2, z2)
CREATE ELEMENT (E1, POINT, x0, y0, z0)
CREATE ELEMENT (E1, POLYGON,  n); DATA(x1,y1,z1, ..., xn,yn,zn)
CREATE ELEMENT (E1, POLYLINE,  n); DATA(x1,y1,z1, ..., xn,yn,zn)
CREATE ELEMENT (E1, QUAD_STRIP, n); DATA(x1,y1,z1, ..., xn,yn,zn)
DATA (xi, yi, zi, xj, yj, zj, ... , xn, yn, zn)
CREATE CONTOUR_LINE_ARC(C1, E1 || SELECTION)
```

Grid Functions

```
DEFORM GRID (G1, ALIGNED, Y, double, double)
EXPORT GRID (SELECTION, FILENAME, "filename")
REVOLVE ELEMENT (G1, SELECTION, GRID_ROT_CONTOUR, w1, w2, nNodes)
ROTATE GRID (G1, cx, cy, cz, wx, wy, wz)
SET POINT (G1, ix, iy, X || Y || Z, double)
SET POINT (G1, ix, iy, x, y, z)
```

Rohdaten Eigenschaften

```
SET PROPERTY(E1, NORMALVECTOR, x1, y1, z1)
SET PROPERTY(E1, NORMALDIRECTION, 1)
SET PROPERTY(E1, INUSE, bool)
```

Rohdaten Verwalten

```
CLEAR ELEMENT (E1 || SELECTION)
CONCATENATE ELEMENTS (E1, RING, x0, y0, n)
GROUP ELEMENTS (SELECTION)
GROUP ELEMENTS (ALL)
SELECT CONTOUR (C1)
SELECT ELEMENT (E1)
TRANSFORM ELEMENTS (E1, CONTOUR)
TRANSFORM ELEMENTS (E1, LINE_SEGMENTS, dL)
UNGROUP ELEMENTS (ALL)
```

Rohdaten Bewegen

```
MOVE ELEMENT (E1 || SELECTION, MOVE_MATRIX, O1 || SELECTION)
TRANSLATE GRID (G1 || SELECTION, ABSOLUTE || RELATIVE, dx,dy,dz)
```

Primitives

Primitives Erzeugen

```
CREATE OBJECT (O1, CONE, x0, y0, z0, wx, wy, wz, R, r, dz)
CREATE OBJECT (O1, CONE, x1, y1, z1, x2, y2, z2, r1, r2)
```

```
CREATE OBJECT (O1, CUBOID, x0, y0, z0, wx, wy, wz, dx, dy, dz)
CREATE OBJECT (O1, CUBOID, x1, y1, z1, x2, y2, z2)
CREATE OBJECT (O1, CYLINDER, x0, y0, z0, wx, wy, wz, R, dz)
CREATE OBJECT (O1, CYLINDER, x1, y1, z1, x2, y2, z2, R)
CREATE OBJECT (O1, PARTICLE_SPHERE, x0, y0, z0, R)
CREATE OBJECT (O1, PLANE, nx0, ny0, nz0, nx1, ny1, nz1)
CREATE OBJECT (O1, PRISM, E1, EXTRUSION, dz)
CREATE OBJECT (O1, PRISM_LINE_ARC, E1 || SELECTION, EXTRUSION,dz)
CREATE OBJECT (O1, PRISM_QUAD_STRIP, E1, EXTRUSION, dz)
CREATE OBJECT (O1, SPHERE, x0, y0, z0, R)
CREATE OBJECT (O1, TORUS, R, r)
CREATE OBJECT (O1, TORUS_SEGMENT, R, r, phi)
CREATE OBJECT (O1, TUBE, x0, y0, z0, wx, wy, wz, Ra, Ri, dz)
CREATE OBJECT (O1, TUBE_SEGMENT, x1,y1,z1,x2,y2,z2,Ra,Ri,phi,ws)
CREATE OBJECT (O1, TUBE_SURFACE, x0, y0, z0, wx, wy, wz, R, dz)
```

Fixpunkte Erzeugen

```
CREATE FIXPOINT (F1, x0, y0, z0)
CREATE FIXPOINT (F1, SELECTION)
```

Spezielle Objekte

```
REVOLVE SECTION (O1, SELECTION)
TWIST OBJECT (O1 || SELECTION, Tx, Ty, Tz, dz)
```

Primitives Befestigen

```
ATTACH OBJECT (SELECTION, TRUE || FALSE)
ATTACH OBJECTGROUP (oNr, LAST_GROUP_NR, x, y, z, x2, y2, z2, kc)
ATTACH OBJGRP_OBJGRP (oNr1, oNr2, mode)
```

Primitives Verwalten

```
CLEAR OBJECT (O1 || SELECTION)
DUPLICATE OBJECT (O1 || SELECTION || ALL, INPLACE || bool)
RESET ANGLE (ALL)
RESET F_EXT (ALL)
```

Primitives Selektieren

```
SELECT OBJECT (ALL)
SELECT OBJECT (LAST_OBJECT - n)
SELECT OBJECT (O1) || SELECT MICROBE(O1)
SELECT OBJECT (POINT, x1, y1, z1)
SELECT OBJECT (POINT, ALL, x1, y1, z1)
SELECT OBJECT (RECT, XY, x1, x2, y1, y2)
SELECT OBJECT (SPACE, x1, y1, z1, x2, y2, z2)
SELECT OBJECT (COLOR_STD, index)
SELECT OBJECT (COLOR_RGB, red, green, blue)
SELECT FIXPOINT (F1)
```

Primitive-Eigenschaften Setzen

```
CREATE IACT_RULE(O1, O2, bool)
CREATE IACT_RULE(O1, O2, bool, mode1, mode2)
CREATE IACT_RULE(SELECTION, bool, mode1, mode2)
SET PROPERTY(O1, ANGULAR_VELOCITY, {X|Y|Z}, double)
SET PROPERTY(O1, BEVEL, ROUND || FACET, double)
SET PROPERTY(O1, C_INTERACT_LIN, 0.0001)
SET PROPERTY(O1, C_INTERACT_QUAD, 0.0001)
SET PROPERTY(O1, COLOR_STD, 4)
SET PROPERTY(O1, COLOR_RGB, 255, 0, 0)
SET PROPERTY(O1, INTERACT_CONTROLPOINT, x1, y1, z1, TRUE)
SET PROPERTY(O1, DENSITY, 7.8)
SET PROPERTY(O1, END_OF_WIRE, TRUE);
SET PROPERTY(O1, FORCE_EXT, {X|Y|Z}, double)
SET PROPERTY(O1, FRICTION_UNILATERAL, double)
SET PROPERTY(O1, GLUE, TRUE)
```

```
SET PROPERTY(O1, GROUP_NR, LAST_GROUP_NR)
SET PROPERTY(O1, INTERACT_DIRECTION, RADIAL_ONLY || AXIAL_ONLY ||
   ALL_DIRECTIONS)
SET PROPERTY(O1, INTERACT_METHOD, ELASTIC, 10)
SET PROPERTY(O1, INTERACT_MODE, {ACTIVE,PASSIVE,NO_INTERACTION})
SET PROPERTY(O1, LONG_CONTOUR_OBJECT, TRUE)
SET PROPERTY(O1, MASS, double)
SET PROPERTY(O1, MOMENT_FORCE_EXT, nx, ny, nz, F)
SET PROPERTY(O1, MOMENT_INERTIA, Ix, Iy, Iz)
SET PROPERTY(O1, MOMENT_INERTIA, FACTOR, double)
SET PROPERTY(O1, NAME, "objectname")
SET PROPERTY(O1, ROTATION_LOCKED, X|Y|Z, TRUE|FALSE)
SET PROPERTY(O1, SIM_MEMBER, FALSE)
SET PROPERTY(O1, SUPERGROUP_NR, LAST_SUPERGROUP_NR)
SET PROPERTY(O1, TRANSPARENCY, integer)
SET PROPERTY(O1, VELOCITY, {X|Y|Z}, double)
SET PROPERTY(O1, VISCOSITY, TRUE)
SET PROPERTY(O1, VISIBILITY, FALSE)
SET PROPERTY(O1, WIREFRAME, TRUE)
```

Primitives Bewegen

```
MOVE OBJECT (O1 || SELECTION, ABSOLUTE || RELATIVE, x0, y0, z0,
   wx, wy, wz)
ROTATE OBJECT (O1, cx, cy, cz, wx, wy, wz)
TRANSLATE OBJECT(O1 || SELECTION, ABSOLUTE || RELATIVE, dx,dy,dz)
```

Primitive-Gruppen und Clusters

Seile

```
CREATE CABLE_BLOCKMODEL (
   x1,y1,z1, x2,y2,z2,
   n0, wd0,
   ro, K1, K2, K3, C1, C2)
```

Biegsames Rohr

```
CREATE TUBE_LINE (x1, y1, z1, x2, y2, z2,
   n0, da, di, ro, E, G, C1, C2, mode)
```

Biegsame Drähte

```
CREATE WIRE_LINE(x1, y1, z1, x2, y2, z2,
   n0, wd0, ro, E, G, C1, C2, mode)
CREATE WIRE_RING (R, r, n, ro, E, G, C1, C2,
   DEFAULT || EXCL120 || CTR120)
```

Federn

```
CREATE SPRING (HELICAL, TENSION || COMPRESSION, STANDARD,
   D1, L0, Ln, d, ro, wd, C1, C2, F0, Fn)
CREATE SPRING (LEAF, STANDARD, dL, dw, dh, n0, ro, E, G, C1, C2,
   mode)
```

Objektgruppe Bewegen

```
ROTATE OBJECTGROUP(O1, cx, cy, cz, wx, wy, wz)
ROTATE OBJECTSUPERGROUP(O1, cx, cy, cz, wx, wy, wz)
TRANSLATE OBJECTGROUP(O1, ABSOLUTE || RELATIVE, dx, dy, dz)
TRANSLATE OBJECTSUPERGROUP(O1, ABSOLUTE || RELATIVE, dx, dy, dz)
```

Allgemeine Funktionen

```
JOIN OBJECTGROUP (O1, Emodul)
JOIN OBJECTSUPERGROUP (O1, Emodul)
```

Objektgruppen-Eigenschaften Setzen

```
SET GROUP_PROPERTY(O1, VELOCITY, {X|Y|Z}, vAbs)
SET GROUP_PROPERTY(O1, SUPERGROUP_NR, LAST_SUPERGROUP_NR)
SET GROUP_PROPERTY(O1, C_DAMPING, BENDING, double)
```

```
SET GROUP_PROPERTY(O1, C_DAMPING, NORMAL, double)
SET GROUP_PROPERTY(O1, C_DAMPING, TORSION, double)
SET GROUP_PROPERTY(O1, C_LINK, BENDING, double)
SET GROUP_PROPERTY(O1, C_LINK, NORMAL, double)
SET GROUP_PROPERTY(O1, C_LINK, TORSION, double)
SET GROUP_PROPERTY(O1, DENSITY, double)
SET GROUP_PROPERTY(O1, GLUE, c, RMax, TRUE)
SET GROUP_PROPERTY(O1, VISCOSITY, C, N, T, cA, cD, dMax, TRUE)
SET GROUP_PROPERTY(O1, ROTATION_LOCKED, X|Y|Z, TRUE || FALSE,
   ALL || FIRST_LAST)
SET GROUP_PROPERTY(O1, COLOR_STD, 4)
SET GROUP_PROPERTY(O1, COLOR_RGB, 255, 0, 0)
SET GROUP_PROPERTY(O1, C_LINK, STRAIN_LIMIT,
   PERCENT || ABSOLUTE, double
SET GROUP_PROPERTY(O1, C_LINK, STRENGTH_CALC, mode, value, fac)
SET GROUP_PROPERTY(O1, C_LINK, STRENGTH_ULTIMATE, mode,value,fac)
SET GROUP_PROPERTY(O1, C_LINK, UNLIMITED, bool)
SET GROUP_PROPERTY(O1, C_LINK, OVERLOAD_ACTION, action)
SET GROUP_PROPERTY(O1, C_LINK, MATERIAL_MODEL, modelNr)
SET GROUP_PROPERTY(O1, C_LINK, CTR120, K_ctr, K_r)
SET GROUP_PROPERTY(O1, C_INTERACT_LIN, double)
SET GROUP_PROPERTY(O1, C_INTERACT_QUAD, double)
SET GROUP_PROPERTY(O1, FORCE_EXT, {X|Y|Z}, double)
SET GROUP_PROPERTY(O1, INTERACT_MODE, {ACTIVE,PASSIVE,..})
SET GROUP_PROPERTY(O1, SIM_MEMBER, FALSE)
SET GROUP_PROPERTY(O1, YIELD_MODEL, modelNr)
SET STATE (BENDING_STRENGTH, ON || OFF, SUPERGROUP_NR,
   LAST_SUPERGROUP_NR)
SET VALUE (E_MODUL= double, SUPERGROUP_NR, LAST_SUPERGROUP_NR, f)
SET VALUE (BENDING_STRENGTH = double, SUPERGROUP_NR,
   LAST_SUPERGROUP_NR, f)
```

Group- and Supergroup Operationen

```
SET VALUE (NEW_GROUP_NR)
SET VALUE (NEW_SUPERGROUP_NR)
```
eine neue Gruppen-Nr erzeugen

```
SET PROPERTY (O1, GROUP_NR, LAST_GROUP_NR)    (*)
SET PROPERTY (O1, SUPERGROUP_NR, LAST_SUPERGROUP_NR)
```
einzelne Objekte zu einer Gruppe hinzufügen

```
SET GROUP_PROPERTY (O1, SUPERGROUP_NR, LAST_SUPERGROUP_NR)   (**)
SET GROUP_PROPERTY (LAST_GROUP_NR, SUPERGROUP_NR,
                    LAST_SUPERGROUP_NR)
```
eine Gruppe von Objekten zu einer Supergroup hinzufügen

(*) O1 ist hier ein bestimmtes einzelnes Objekt
(**) O1 ist hier ein Repräsentant einer Objektgruppe

Links

```
CREATE LINK (K1, NORMAL, O1, O2, E1)
CREATE LINK (K1, NORMAL || BENDING || TORSION, O1, O2, x0,y0,z0)
CREATE LINK (K1, NORMAL, O1, F1)
CREATE LINK (K1, NORMAL || BENDING || TORSION, SELECTION)
CREATE LINK (K1, NORMAL, AUTOMATIC, E1)
CREATE LINK (K1, NORMAL, AUTOMATIC, E1, O1)
CREATE LINK (K1, NORMAL, AUTOMATIC, CTR120, E1)
```
Links Erzeugen

```
CREATE LINK (K1, NORMAL, CTR120, O1, O2)
CREATE LINK (K1, U_BREMSE, O1, O2, x1,y1,z1, x2,y2,z2)

SELECT LINK (K1)
SELECT LINK (x0, y0, z0)

SET PROPERTY(K1, ANGLE, {X|Y|Z}, double)
SET PROPERTY(K1, C_DAMPING, double)
SET PROPERTY(K1, C_LINK, double)
SET PROPERTY(K1, MOMENT_FRICTION, double)

SET VALUE (C_LINK = double, linktype, OBJECTNAME, "objectname")
```

Control Systems

Punkt Kurven

```
CREATE POINT_CURVE (
   O1, SIMPLE, varX, varY, wx, wy, wz, mode, activation, nPoints,
   x1, y1, ... , xn, yn)
CREATE POINT_CURVE (
   O1, FILENAME, "filename", varX, varY, wx, wy, wz, mode,
activation)
```

Kontrollsysteme

```
CREATE CONTROLSYSTEM_AUTO(o1,v1,c1,vl1,vl2, o2,v2,c2,vl3,vl4,dr)
```

Allg. Verwaltungsfunktionen

```
BEGIN SCRIPT scriptname || BEGIN MACRO scriptname
CLEAR ALL
DESELECT ALL
```

Globale Eigenschaften

```
CREATE FIELD(Gravitation, nx, ny, nz, b)
SET STATE (FRICTION, ON || OFF)
SET STATE (GLUE, ON || OFF)
SET STATE (VISCOSITY, ON || OFF)
SET VALUE (C_FRICTION_DYNAMIC = double)
SET VALUE (C_FRICTION_STATIC = double)
SET VALUE (C_GLUE = double)
SET VALUE (C_VISCOSITY = double)
SET VALUE (MATERIAL_PARAM, Index, A, B, C, D, E, F, n1, n2, n3)
SET VALUE (MODEL_SCALE, SCREEN_WORLD = value, REAL_WORLD = value
SET VALUE (MODEL_SIZE_FAC_OPENGL = double)
SET VALUE (NEW_GROUP_NR)
SET VALUE (NEW_SUPERGROUP_NR)
SET VALUE (TIMESTEP_MAX = double)
SET VALUE (VISCOSITY_RANGE = double)
```

Simulations-Steuerung

```
SET PREFERENCE (IACT_CYCLE_CALC_PERIOD, int)
SET PREFERENCE (IACT_DEPTH_LIMIT_ACTIVE, TRUE)
SET PREFERENCE (IACT_DEPTH_LIMIT_MSG, TRUE)
SET PREFERENCE (IACT_DEPTH_LIMIT_VALUE, double)
SET PREFERENCE (IACT_OBJSIZE_SECURITY_FAC, double)
```

Programm Steuerung

Bedingte Ausführung

```
DO IF (boolean condition)
  statementList
END IF
```

Schleifen

```
LOOP FOR (K, i1, i2, step)
  statementList
END FOR
```

Objekt Referenz festlegen

```
FIND OBJECT (O1, x0, y0, z0)
SET VALUE (O1 = SELECTION)
```

Modul Cable

Zopf

```
CREATE BRAID (
   x1, y1, z1, x2, y2, z2,
   SL, D, nW, nE, d,
   ro, E, G, C1, C2, colorIdx, iActMethodIdx, iActModeIdx,
   matIndex)
```

Litzenzopf

```
CREATE BRAID_LITZE (
   x1, y1, z1, x2, y2, z2,
   SL, D, nL, SL0, SR0,
   d0,n0,e0,
   d1,n1,e1,
   d2,n2,e2,
   d3,n3,e3,
   d, ro, E, G, C1, C2, colorIdx, iActMethodIdx, iActModeIdx,
   matIndex)
```

Seil

```
CREATE CABLE (
   x1, y1, z1, x2, y2, z2,
   SL_L0, SR_L0, n0_L0, e0_L0, d0_L0,
                 n1_L0, e1_L0, d1_L0,
                 n2_L0, e2_L0, d2_L0,
                 n3_L0, e3_L0, d3_L0,
   SL_L1, SR_L1, n0_L1, e0_L1, d0_L1,
                 n1_L1, e1_L1, d1_L1,
                 n2_L1, e2_L1, d2_L1,
                 n3_L1, e3_L1, d3_L1,
   SL_L2, SR_L2, n0_L2, e0_L2, d0_L2,
                 n1_L2, e1_L2, d1_L2,
                 n2_L2, e2_L2, d2_L2,
                 n3_L2, e3_L2, d3_L2,
   SL_S1, SR_S1, nS1, H_diam_S1,
   SL_S2, SR_S2, nS2, H_diam_S2,
   ro, E, G, C1, C2, colorIdx, iActMethodIdx, iActModeIdx,
   matIndex)
```

Litze

```
CREATE STRAND (x1, y1, z1, x2, y2, z2,
   n0, d0, e0, n1, d1, e1, n2, d2, e2,
   SL, SR,
   ro, E, G, C1, C2)
```

Komplexe Litze

```
CREATE STRAND_COMPLEX (x1, y1, z1, x2, y2, z2,
   SL, SR, n0, e0_SL, wd0,
   n1, e1_SL, wd1, D1, phi1,
   n2, e2_SL, wd2, D2, phi2,
   n3, e3_SL, wd3, D3, phi3,
   n4, e4_SL, wd4, D4, phi4,
   n5, e5_SL, wd5, D5, phi5,
   n6, e6_SL, wd6, D6, phi6,
   ro, E, G, C1, C2, colorIdx, iActMethodIdx, iActModeIdx,
   matIndex)
```

Drahthelix

```
CREATE WIRE_HELIX (x1, y1, z1, x2, y2, z2,
   SL, SR, e0_SL, wd0, D, phi,
   ro, E, G, C1, C2, DEFAULT || EXCL120)
```

Funtionen

```
SET VALUE (CABLE_CORRECTIONS, HELIX, sNr, wNr, double)
SET VALUE (CABLE_CORRECTIONS, ANGLE, sNr, TRUE|FALSE)
```

Modul Chain

```
CREATE CHAIN (
   typeNr,e1,nK,nA,o1,o2,o3,o4,nB,o5,o6,o7,o8,b1,b2,S,C1,C2)
```

Modul Profile

Profile Erzeugen

```
CREATE PROFILE (TYPE_STRIPES, nStripes, nSections,r o, E, G,
   C1, C2, colIdx, iActMethodIdx, iActModeIdx, matIndex,
   x11, y11, z11, x12, y12, z12,
   ...
   x91, y91, z91, x92, y92, z92,
   Lz1, n1, ..., Lz9, n9,
   nLinks,
   k1x, k1y, ..., k18x, k18y)
CREATE PROFILE (TYPE_SHEET_METAL, d, L, nE, nL, ro, E)
CREATE PROFILE (TYPE_UNIFORM_PLATE, x1, x1, z1, x2, y2, z2,
   nx, ny, nz, ro, E, G, C1, C2, colIdx,
   iActMethodIdx, iActModeIdx, matIndex)
SWEEP CROSSSECTION (O1, SELECTION, POLYLINE, CIRCLE, R)
SWEEP CROSSSECTION (O1, SELECTION, LINE_ARC, CIRCLE, R)
```

Funktionen

```
ENHANCE RESOLUTION (x1,y1,z1, x2,y2,z2, eL, E, G) ..,eMode,aMode)
FRAGMENT OBJECT (O1, CUBOID, dx, dy, dz, minSize)
```

Modul Particles

```
CREATE CLUSTER (PARTICLE_SPHERE, DENSE || CUBIC, x0, y0, z0, nx,
ny, nz, R1, R2)
CREATE FIBRE   (x1, y1, z1, x2, y2, z2, n0, d0, ro, E, G, C1, C2)
```

Konstanten

einige sonar script Anweisungen benutzen die Parameter ‘iActMethodIdx’ und ‘iActModeIdx’. Diese Parameter (integer) haben in Bezug auf die Bezeichnungen wie sie in den Dialogen benutzt werden die folgende Bedeutung:

0: CUSTOM
1: EP_100_0
2: EP_90_10
3: EP_75_25 (Elastic 75%, Plastic 25%)
4: EP_50_50
5: EP_25_75
6: EP_10_90
7: EP_0_100

iActMethodIdx (interaction method)

0: NO_INTERACTION
1: ACTIVE
2: PASSIVE

iActModeIdx (interaction mode)

positive Werte: 1, TRUE, YES, ON
negative Werte: 0, FALSE, NO, OFF

Bool'sche Werte

X, Y, Z

kartesische Koordinaten

CTR120, EXCL120, NNN, NNNN, NBT, NNB

Spezifizierung von Link-Kombinationen

Kontrollsystem Sprache

Ueberblick

Nebst der Anwendung als Makrosprache wird ‘sonar script’ auch als Kontrollsystemsprache zur kontinuierlichen Steuerung von Objekten während einer laufenden Simulation eingesetzt. Wäre Objekt (o1) beispielsweise ein im Schwerpunkt drehbar gelagertes Objekt, dann würde das folgende Kontrollsystem dafür sorgen, dass die Drehzahl von Objekt ‘o1’ zunehmend ansteigt bis eine gewisse Drehzahl erreicht wird. Anschliessend würde das Objekt kontrolliert auf der gegebenen Solldrehzahl gehalten.

Beispiel

```
CONTROLSYSTEM speed_control
DO IF (OMG(o1) < 1.0E-5)                              (A)
  SET VALUE (MOMENT_FORCE_EXT.z(o1) = 1.0E-8)         (B)
END IF
DO IF (OMG(o1) > 1.0E-5)
  SET VALUE (MOMENT_FORCE_EXT.z(o1) = -1.0E-8)
END IF
-- end of control system
```

Bereits aus diesem einfachen Beispiel geht hervor, dass es grundsätzlich zwei Arten von Anweisungen gibt.

1. Es gibt sog. bedingte Anweisungen welche als Resultat einen bool’schen Wert (true/false) liefern und dafür sorgen, dass nachfolgende Anweisungen nur unter gewissen Bedingungen ausgeführt werden (A).
2. Und es gibt sog. Zuweisungen welche einer bestimmten Objektvariablen einen berechneten oder gegebenen Wert zuweisen (B).

In den verwendeten Ausdrücken dürfen alle geometrischen und physikalischen Variablen vorhandener Objekte entweder verwendet oder neu berechnet und gesetzt werden.

Grammatik

Die Vielfalt an möglichen Anweisungen in einem Kontrollsystem lässt es nicht zu, die Summe aller möglichen Anweisungen in der geschlossenen Form anzugeben, wie das bei Makros der Fall ist. Mit den vorhandenen geometrischen und physikalischen Variablen sämtlicher Objekte lassen sich letztlich endlos viele Varianten von Formeln bilden. Wir geben deshalb im folgenden die Grammatik der Kontrollsystem-Sprache in einer möglichen Beschreibungsforman an. Es handelt sich um die Darstellung der sog. Backus Naur Form der Syntax.

```
<Controlsystem> ::= <Statement> <CR> <Controlsystem> | <NUL>
<CR>            ::= carriage return character (ASCII 13)
<NUL>           ::= end of Controlsystem
<Statement>     ::= <Command> <Qualifier>
<Statement>     ::= <Command> <Qualifier> ( <Assignment> )
<Statement>     ::= <Command> <Qualifier> ( <BoolExpression> )
<Command>       ::= <Identifier>
<Assignment>    ::= <Variable> = <Expression>
<Expression>    ::= <Term> | ( <Term> +|- <Term> )
<Term>          ::= <Factor> | ( <Factor> *|/ <Factor> )
<Factor>        ::= <Number> | <Variable> | ( <Expression> )
<Factor>        ::= <Function> ( <Expression> )
<BoolExpression>::= <Expression> <Skal.Operand> <Expression>
<BoolExpression ::= <BoolExpression> <Bool.Op.> <BoolExpression>
<Variable>      ::= <Skal.Variable> | <Vect.Variable>
<Skal.Variable> ::= <Identifier> ( <Obj.Variable> )
<Vect.Variable> ::= <Identifier> . <Coord> ( <Obj.Variable> )
<Obj.Variable>  ::= O|K|A|P <Pos.Integer>
<LoopVar>       ::= character
<Const>         ::= <Integer> | <Decimal>
<Coord>         ::= X | Y | Z
<Function>      ::= SIN | COS | TAN | ASIN | ACOS | ATAN | SINH |
                    COSH | TANH | ABS | SQR | SQRT | EXP | LOG |
                    LOG10 | ODD
<Operation>     ::= + | - | *| /
<Skal.Operand>  ::= < | <= | == | >= | >
<Bool.Operand>  ::= AND | OR | NOT
<Identifier>    ::= list of words
<Qualifier>     ::= list of words
<Number>        ::= <Decimal> | <Integer>
<Decimal>       ::= 123.45678.. | 1.2345678E+02
<Integer>       ::= positiv or negativ integral number
<Pos.Integer>   ::= positiv integral number (1,2,3,..)
```

Reservierte Worte

Command

ALARM, ATTACH, BEGIN, CALC, CALCULATE, CHANGE, CLEAR, CLOSE, CONCATENATE, CONTROLSYSTEM, COPY, CREATE, CUT, DEFORM, DESELECT, DISTORT, DO, DRAG, DUPLICATE, END, ENHANCE, EXECUTE, EXPORT, FIND, FRAGMENT, GROUP, IMPORT, JOIN, LOAD, LOOP, MACRO, MIRROR, MOVE, NEW, OPEN, PASTE, QUIT, RESET, RETURN, REVOLVE, ROTATE, RUN, SAVE, SCRIPT, SELECT, SET, SIZE, SKEW, START, STEP, STOP, STRETCH, SWEEP, TRANSFORM, TRANSLATE, TWIST, UNGROUP

Qualifier

ALL, ANGLE, BRAID, BRAID_LITZE, CABLE, CABLE_BLOCKMODEL, CHAIN, CLUSTER, COLLECTION_LINE_ARC, CONTOUR, CONTOUR_LINE_ARC, CONTROLSYSTEM_AUTO, CROSSSECTION, DIALOG, DIAMETER, ELEMENT, ELEMENTS, FIBRE, FIELD, FILE, FIXPOINT, FIXPOINTS, FOR, FORMULA, GRAPHIC, GRID, GROUP_PROPERTY, IACT_RULE, IF, LINK, MACRO, MODEL, OBJECT, OBJECTGROUP, OBJECTS, OBJECTSUPERGROUP, OBJGRP_OBJGRP, POINT, POINT_CURVE, POLYLINE, POS, POS_X, POS_Y, POSITION, PREFERENCE, PROFILE, PROGRAM, PROPERTY, PROPULSION, RADIUS, RECORD, RECORDING, RESOLUTION, SECTION, SIMULATION, SPRING, SPRING_SPIRAL, STATE, STRAND, STRAND_COMPLEX, SUPERGROUP_PROPERTY, TUBE_LINE, VALUE, WHILE, WINDOW, WIRE_HELIX, WIRE_LINE, WIRE_RING

Parameter

ABSOLUTE, ACTIVE, ALIGNED, ALL, ANGLE, ANGLE_MAX, ANGLE_MIN, ANGLE_ROTATION, ANGULAR_VELOCITY, ARC, ARC2, ARROW, ASK, AUTOMATIC, AUTONOM, AXIAL_ONLY, BENDING, BENDING_STRENGTH, BEVEL, BREAK_DRAW_OUT, BREAKUP, BUTTON, C, C_DAMPING, C_FRICTION_DYNAMIC, C_FRICTION_STATIC, C_GLUE, C_INTERACT_LIN, C_INTERACT_QUAD, C_LINK, C_VISCOSITY, CABLE_CORRECTIONS, CANCEL, CIRCLE, CIRCLE, COLOR, COLOR_RGB, COLOR_STD, COLORING, COMPRESSION, CONE, CONE, CONSTANT, CONTOUR, CONTROL, CORNER, CREATE_CABLE, CREATE_FIELD, CREATE_OBJECT, CREATE_PROPULSION, CROSSING, CTLSYSTEM, CTLSYSTEM_COMPLEX, CTLSYSTEM_SIMPLE, CTR120, CUBIC, CUBOID, CYCLE, CYLINDER, DATABASE, DECORATIVE, DEFAULT, DELTA, DENSE, DENSITY, DIAMETER, DIRECTION, DIRECTION_BUTTON_NEG, DIRECTION_BUTTON_POS, DIRECTION_CONTROL, DIRECTION_FUNCTION, DIRECTION_INTERFACE, DIRECTION_MAX, DIRECTION_MIN, DISTR_AUTO, DISTR_FUNCTION, DISTR_REMOVEMARK, DISTR_SETMARK, DISTR_SHIFTMARK, DISTR_STEPIN, DISTR_STEPOUT, DISTR_STOP, DURATION, E_MODUL, EDIT_FIELD, EDIT_LINK, EDIT_OBJECT, EDIT_PROPULSION, ELASTIC, END, END_OF_WIRE, ENDPOINT, EXCL120, EXTERNAL, EXTRUSION, FACET, FACTOR, FALSE, FILENAME, FIRST_LAST, FORCE, FORCE_EXT, FREE, FRICTION, FRICTION_UNILATERAL, FUNCTION, FUNCTIONAL, GEAR, GLOBAL, GLUE, GLUE_RANGE_FACTOR, GRAVITATION, GRID_ROT_CONTOUR, GROUP_NR, GROUPED, HELICAL, HELIX, HORIZONTAL, IACT_CYCLE_CALC_PERIOD, IACT_DEPTH_LIMIT_ACTIVE, IACT_DEPTH_LIMIT_GRID_ACTIVE, IACT_DEPTH_LIMIT_GRID_VALUE, IACT_DEPTH_LIMIT_MSG, IACT_DEPTH_LIMIT_VALUE, IACT_OBJSIZE_SECURITY_FAC, IMPORT_POLYLINE, INPLACE, INTERACT, INTERACT_CONTROLPOINT, INTERACT_DIRECTION, INTERACT_MEMBER, INTERACT_METHOD, INTERACT_MODE, INTERACT_RULES, INTERACTIVE, INTERFACE, INTERNAL, INUSE, JOYSTICK, KEYBOARD, LAST_GROUP_NR, LAST_OBJECT, LAST_SUPERGROUP_NR, LEAF, LENGTH, LINE, LINE_ARC, LINE_SEGMENTS, LINEAR, LINK, LOCAL, LONG_CONTOUR_OBJECT, MASS, MATERIAL, MATERIAL_MODEL, MATERIAL_PARAM, MAX, MIDPOINT, MODE, MODEL_SCALE, MODEL_SIZE_FAC_OPENGL, MOMENT_BENDING, MOMENT_FORCE_EXT, MOMENT_FRICTION, MOMENT_INERTIA, MOUSE, MOVE_MATRIX, NAME, NEGATIVE, NEUTRAL, NEW_GROUP_NR, NEW_SUPERGROUP_NR, NO, NO_ACTION, NO_INTERACTION, NONE, NORM_DIRECTION, NORM_VECTOR, NORMAL, NORMALDIRECTION, NORMALVECTOR, NUMBER_AUTO, NUMBER_ELEMENTS, NUMBER_FIX, OBJECTNAME, OFF, OK, ON, OVERLOAD_ACTION, PARTICLE_SPHERE, PASSIVE, PERCENT, PI, PLANE, PLASTIC, POINT, POLYGON, POLYLINE, POSITION, POSITIVE, PRISM, PRISM_LINE_ARC, PRISM_QUAD_STRIP, QUAD_STRIP, RADIAL_ONLY, REAL_WORLD, RECT, RELATIVE, RESTRICTION, RING, ROTATION, ROTATION_LOCKED, ROUND, SCREEN_LAYOUT, SCREEN_WORLD,

SELECTION, SHAPE, SIGNAL, SIM_MEMBER, SIMPLE, SINGLE, SPACE, SPECIFIC, SPHERE, STANDARD, START, STATE, STRAIN_LIMIT, STRENGTH_CALC, STRENGTH_MIN, STRENGTH_ULTIMATE, SUPERGROUP_NR, SYMBOL, SYMBOL_LENGTH, T, TENSION, THICKNESS, TIME, TIMESTEP_MAX, TORSION, TORUS, TORUS_SEGMENT, TORUS_SEGMENT_2, TRANSPARENCY, TRIGGER, TRIGGER_BUTTON, TRIGGER_CONTROL, TRIGGER_COUNTER, TRIGGER_DELAY, TRIGGER_FUNCTION, TRIGGER_INTERFACE, TRIGGER_MAX, TRIGGER_STATE, TRIGGER_TIME, TRIGGER-MODE, TRUE, TUBE, TUBE_SEGMENT, TUBE_SURFACE, TYPE, TYPE_SHEET_METAL, TYPE_STRIPES, TYPE_UNIFORM_PLATE, U_BREMSE, UNLIMITED, UNUSED, VALUE, VALUE_BUTTON_NEG, VALUE_BUTTON_POS, VALUE_CONTROL, VALUE_FORCE, VALUE_FUNCTION, VALUE_INTERFACE, VALUE_MAX, VALUE_MIN, VELOCITY, VERTICAL, VISCOSITY, VISCOSITY_RANGE, VISIBILITY, VOLUME, WIREFRAME, X, XY, Y, YES, Z

Physical Variables und ihre Abkürzungen

A, ACCELERATION, AM, ANG, ANG_ROT, ANGLE, ANGLE_ROTATION, ANGULAR_ACCELERATION, ANGULAR_MOMENTUM, ANGULAR_VELOCITY, BETA, C, CENTER_MASS, CM, CYCLE, D, DIR, DIRECTION, DISTANCE, DUR, DURATION, E_KIN, E_KIN_TOT, E_ROT, E_ROT_TOT, E_TOT, EKIN, EKIN_TOT, EROT, EROT_TOT, ETOT, F, F_EXT, F_IACT, FORCE, FORCE_EXT, FORCE_INTERACTION, LEN, LENGTH, M, M_EXT, M_FRICTION, M_TOT, MOMENT_FORCE, MOMENT_FORCE_EXT, MOMENT_FRICTION, MOMENTUM, MOMENTUM_TOT, OMG, P, POS, POSITION, T, TIME, TRG_COUNTER, TRG_DELAY, TRG_STATE, TRG_TIME, TRIGGER_COUNTER, TRIGGER_DELAY, TRIGGER_STATE, TRIGGER_TIME, V, VELOCITY

Koordinaten

X, Y, Z