

# Part V:
# MAAT Hydro's STC Scripting Reference Guide
# Rev. 7.10

© Sistre January 2015

# TABLE OF CONTENTS


1 General rules and preliminary guidelines: 3
2 Developing and testing STC scripts: 8
3 STC Scripting Reference Guide: 9
3.1 STC Variable Declarators: 10
3.1.1: Heelpoint: 10
3.1.2: Curve: 15
3.1.3: Area: 19
3.1.4: Lever: 20
3.1.5: InputLever: 21
3.1.6: Data: 22
3.1.7: InputData: 25
3.2 STC Commands: 26
3.2.1: Abort: 26
3.2.2: AddCSVLineItem: 27
3.2.3: AddList: 31
3.2.4: AreaRange: 33
3.2.5: CalculateWindageData: 34
3.2.6: DefineCargoHeel: 35
3.2.7: DefineSubmersionRange: 36
3.2.8: DefineSwellRange: 37
3.2.9: Else: 38
3.2.10: EndCSVTable: 39
3.2.11: EndIf: 40
3.2.12: For: 41
3.2.13: HeelRange: 43
3.2.14: If: 44
3.2.15 LeverRange: 45
3.2.16: List: 46
3.2.17: ListTab: 47
3.2.18: Maat Stability Criterion: 49
3.2.19: NewCSVLine: 50
3.2.20: Next: 51
3.2.21: Plot: 52
3.2.22: PlotGrid: 53
3.2.23: Print: 54
3.2.24: PrintData: 56
3.2.25: RestrainDamageZone: 57
3.2.26: ReportDamageData: 58
3.2.27: SelectDamageFrom: 59
3.2.28: SetBalanceHeel: 60
3.2.29: SetDamageIcon: 61
3.2.30: SetDamageProperty: 62
3.2.31: SetIndexCoefficients: 64
3.2.32: StartCSVTable: 65
3.2.33: Test: 66
3.2.34: Unit: 68
3.2.35: UserPrompt: 71
3.2.36: WindGradient: 72
3.2 Dimensional Mnemonics: 75
3.3 Hydrostatic Data Mnemonics: 76
3.4.1: Hydrostatic Mnemonics and their corresponding Data: 76
3.4.2: Hydrostatic data and their corresponding mnemonics: 82

# 1 General rules and preliminary guidelines:

Criterion scripting appeared as the only way to face the growing number and complexity of stability criteria, as well as the multiplicity of their local understandings.

Criterion scripting simply consists in writing or modifying a text file with your favorite text editor (Notepad, Wordpad or any alternative editor like NotePad++ …); as this text file has to respect certain rules in order to be understandable by MAAT Hydro, its "txt" extension must be replaced by a "stc" extension (for "STability Criterion") in order to denote this property.

Such a STC file contains all the necessary information for controlling MAAT's Hydrostatic engine in order to, specify the necessary user inputs, calculate / test the requested data and control accurately their output format:

This part of the "Minimal.stc" script is a good introduction to this STC "scripting style":

```
Unit(mm+)
Unit(t)

Heelpoint h0("0°", CONSTANT, 0.0)
Heelpoint h30("30°", CONSTANT, 30.0)
Heelpoint h40("40°", CONSTANT, 40.0)

Curve gz("GZ", GZ)
Curve ox("Heel Axis", CONSTANT, 0.0)

Heelpoint heq("Equilibrium", INTERSECTION+, ox, gz, h0, 1)
Heelpoint hx("GZ Max", MAX, gz)
SetBalanceHeel(heq)

Lever gz30("GZ 30°", GZ, h30)
Lever gzx("GZ Max", GZ, hx)

Curve agz("GZ Area", GZAREA, gz, heq)

Area a030("[0, 30]", ox, gz, 0, h30)
Area a040("[0, 40]", ox, gz, 0, h40)

Area a3040("[30, 40]", ox, gz, h30, h40)
Lever gm0("GM0", GM, 0) // This is just a comment, ignored by the STC interpreter...
//RestoreInitialUnits()// ...and this allows to freeze a whole line for further use...

...
```

Each line of an STC script starts by a ***keyword***, identified above in ***black bold*** characters. Each STC keyword allows defining an action or creating a variable. Actions and variables generally depend on a set of *parameters*, separated by comas and delimited by brackets. The content of the brackets is named the argument.

Moreover, STC files can also include as many user comments as necessary, provided that they are preceded by a double slash "//". This can therefore also be used to freeze script lines in order to conserve them for any further use (see grey characters above).

The line ***Unit*** *(mm+)* defines an action (turns current length units to mm+) whose argument is *(mm+)*.

The line ***Curve*** *gz("GZ", GZ)* creates a variable (representing the GZ curve) named ***gz***, whose argument is *("GZ", GZ)*. This ***gz*** variable name will be available in the rest of the script for further use (i.e. calculating intersections, extrema…) provided that is not used before declaration!

The ***"GZ"*** user name is delimited by quotation marks (free character string excluding quotation marks). This user name will only appear in the outputs for identifying the curve and has no internal use in the script. The *GZ* mnemonic allows to define the curve as the righting arm curve among a list of available mnemonics (GZ, GZAREA, MS, MINFBD…) detailed in the STC Scripting Reference Guide. As all the STC mnemonics are in capital letters, it is then a good habit to avoid using such names for the script variables.

The other expressions appearing in the arguments are numeric constants (that's where unit setting is important) and variable names. For obvious reasons, the variables used in arguments must be created before using them. In addition to this, it is also possible to replace a single variable by a MIN or MAX of variable and/or constants. For example, the following line creates a *cpsz* variable named *"Capsizing"*, representing the smallest angle between the previously calculated capsizing angle *hc* and 50°:

```
Heelpoint cpsz("Capsizing", MIN(hc, 50))
```

Moreover, algebraic expressions can also be used directly, combining variables, constants and operators, provided that a pair of braces delimits them. The following line allows to calculate the '*lw1*' lever defined by IMO 3.2.2.2 from the previously defined variables p, a, z and delta:

```
Lever lw1("Lw1",  {(p*a*z)/delta})//Calculates the 'lw1' lever (IMO
3.2.2.2)
```

The following algebraic expressions illustrate several of these scripting possibilities:

```
{109*k*x1*x2*((r*s)^0.5)}
{twa*hcehlp*(abs(cos(phid))^1.3)}
MAX(0.5, MIN(1.5, {agza/(15.81*(lh^0.5))}))
```

etc...

It is also important to mention here that STC scripting provide the classical If / Then / Else control statements:

```
If(extrem, =, 2.0)// If Damaged Length = Ls
    Data G("G", ABS, {(b11*Jb*Jb/2.0)+(b12*Jb)})

  Else
    If(extrem, =, 0.0)// If damaged area is not at an extremity
```

```
            Data J0("J0", ABS, MIN(J, Jb))
            Data        G("G",        ABS,        {(((b11*J)-b12)*J0*J0/2.0)-
(b11*J0*J0*J0/3.0)+(b12*J*J0)})
          Else// If damaged area is at an extremity
            Data J0("J0", ABS, MIN(J, Jb))
            Data G1("G1", ABS, {(b11*Jb*Jb/2.0)+(b12*Jb)})

            Data       G2("G2",        ABS,        {(((b11*J)-b12)*J0*J0/2.0)-
(b11*J0*J0*J0/3.0)+(b12*J*J0)})
            Data G("G", ABS, {(G2+(G1*J))/2.0})
          EndIf
        EndIf
```

… as well as the For / Next Loops:

```
      For(i, 0, <, imax)// For each damage combination (i starts from 0

        If(i, =, 0)// Restrain Damage zone and set summing sign to pi
accordingly
          RestrainDamageZone(j, n)
          Data sgn("Summing Sign", ABS, {1.0})//
          Data col("c", INT, {col0})
        Else
          If(i, =, 1)

            ...

      if(i, =, 0)
          Print(x1, 5, {col0})//

          Print(x2, 5, {col1})//

          Print(z1, 5, {col2})//

          Print(z2, 5, {col3})//

      EndIf

      Next(1)// Next damage combination (i=i+1)
```

At last, blank characters being ignored by MAAT's STC interpreter, don't hesitate to use them as often as possible for indenting the lines and grouping operations into distinct blocks: A good script layout, combined with numerous comments and smart data names make it clear and easy to understand long after having written it!

An STC script is organized into distinct sections. The reasons of this division are the followings:

- The first line must allow identifying a proper STC script as well as its version and criterion name at opening.
- The units must be specified before specifying the script constants (i.e. testing that GZ is greater than 0.25 is nonsense before specifying that length unit is meter…).
- The user definable data must be defined at the beginning, before being used. It is a good habit to group the user prompts and reminders into this section dedicated to the user interactions.
- The wind strength and gradient must be defined before calculating it's effect on ship's silhouette.

- The plotted and printed variables must be created and calculated before plotting and printing. It is therefore also a good habit to group the plotting statements together after all the data are created and calculated, as well as grouping the data to print and test below. For this reason, plotting and printing statements generally correspond to the last script sections.

The usual script organization is therefore typically the following:

1° Identification header and criterion name
2° Units definition
3° Reminders / prompts to be recalled to the user at runtime
4° User definable inputs definition
5° Criterion and data calculation
6° Diagram's plotting data and layout settings
7° Printout of results and tests.

**Important Remark:**

In order to be usable for Max KG calculations, STC scripts must include at least 1 *Test* statement, MAAT's trial and error loop being unable to locate critical KG values when no test result is available.

Moreover, for obvious reasons, dredge stability criteria need a special header defining hopper's content and properties (see 'DefineCargoHeel' below).

At last, for similar reasons, stability with Water on Deck (Stockholm / Torremolinos, …) also need a special header defining Water on deck's height (see 'DefineSubmersion Range' / 'DefineSwellRange' below).

Such special headers are only used by the appropriate functions.

This "Basic.stc" script, which is MAAT's default stability criterion (located in /ProgData/StabCriteria) will give a short and concrete illustration of these preliminary guidelines:

```
Maat Stability Criterion: "Basic"

Unit(m+)
Unit(t+)

UserPrompt("This Basic Criterion only requires valid Float and Mass
Descriptors")
UserPrompt("As no special line is needed, it is therefore only a default
criterion")
UserPrompt(" ")
UserPrompt("Set Display Heel Range below in 'Min / Max Dsplayed Heel'
fields")
UserPrompt("Démonstration 23-06-2009")

InputData dhmin("Min Displayed Heel", Heel°, 0.0)
InputData dhmax("Max Displayed Heel", Heel°, 50.0)

Heelpoint hmin("Min", MINHEEL)
```

```
Heelpoint h0("0°", CONSTANT, 0.0)
Heelpoint h30("30°", CONSTANT, 30.0)
Heelpoint h40("40°", CONSTANT, 40.0)
Heelpoint hmax("Max", MAXHEEL)

Curve gz("GZ", GZ)
Curve ox("Heel Axis", CONSTANT, 0.0)//Creates a ox axis named "ox" for
further use

Heelpoint heq("Ship Balance", INTERSECTION+, ox, gz, h0)//Finds ship balance
as nearest intersection from h0 and creates a heel point "heq" for further
use
Heelpoint hx("GZ Max", MAX, gz)
SetBalanceHeel(heq)// registers calculated equilibrium

Lever gz30("GZ 30°", GZ, h30)
Lever gzx("GZ Max", GZ, hx)

Curve agz("GZ Area", GZAREA, gz, heq)

Area a030("[0, 30]", ox, gz, h0, h30)
Area a040("[0, 40]", ox, gz, h0, h40)
Area a3040("[30, 40]", ox, gz, h30, h40)

Lever gm0("GM0c", GMTC, h0)

HeelRange(dhmin, dhmax)// Sets stability diagram's heel range to user defined
values (hmin, hmax)
LeverRange(AUTO, AUTO) // Sets automatically lever scale limits to Gzmin and
GZmax
AreaRange(AUTO, AUTO) // Sets automatically GZ Area scale limits to Areamin
and Areamax
PlotGrid(LT, LT)//Sets a medium grid on GZ / GZ Area diagram ("NO"= no, "LT"=
Light, "HI"= full)

Plot(gz)// Plots gz curve on the stability diagram
Plot(agz) // Plots gz area curve on the stability diagram
Plot(hx) // Plots gzmax's heel on the stability diagram

List(heq)// inserts an equilibrium heel data line in the data list
List(hx) // inserts a gzmax heel data line in the data list

ListTab(Heel°, "Heel:", 26, 26) // Lists current heels in column 26 with
"Heel:" title in col. 26
ListTab(TA, "AP Draft:", 38, 36)// idem for AP Draft in col. 38 / 36
ListTab(TF, "FP Draft:", 48, 46) // idem for FP Draft in col. 38 / 36
ListTab(TCB, "TCB:", 53, 54)//etc...
ListTab(VCB, "VCB:", 60, 62)
ListTab(LCB, "LCB:", 69, 70)
ListTab(KN, "KN:", 77, 78)
ListTab(MS, "MS:", 85, 86)
ListTab(GZ, "GZ:", 93, 94)
ListTab(GZArea, "Area:", 102, 102)

Test(hx, >, 25.0, 1, 2)// Tests GZ max heel > 25° & prints result in lower
frame's line 1, col 2
Test(a030, >, 0.055, 1, 38)// idem for a030
Test(a040, >, 0.09, 2, 38)//etc...
Test(a3040, >, 0.03, 3, 38)
Test(gm0, >, 0.150, 1, 81)
Test(gzx, >, 0.250, 2, 81)
Test(gz30, >, 0.200, 3, 81)
```

# *2* Developing and testing STC scripts:

Developing a new script generally never starts from scratch. As a set of reference scripts is provided in the "/ProgData/StabCriteria" directory, you will usually first have to identify the closest one, duplicate and rename it before opening this new text file with your favorite text editor (Notepad, Wordpad, …).

Simultaneously, it may be very helpful to run MAAT and calculate the stability of a ship corresponding to the target criterion. When the stability results will be displayed, you will then be able to select and reselect the criterion you are working on whenever you want, provided that you resaved it. The incidence of your modification will then be tested almost in real time on the displayed results, warning boxes appearing whenever bugs are encountered, to allow you correcting the script immediately in your text editor.

# 3 STC Scripting Reference Guide:

The goal of this section is to provide an accurate description of the available STC keywords, their possible arguments and their typical use. As seen above, these Keywords can be of two kinds:

- STC data declarators.
- STC commands

Section 3.1 therefore lists the available data, Section 3.2 the available STC commands, Section 3.3 the available dimension mnemonics and Section 3.4 the available hydrostatic mnemonics.

## *3.1* STC Variable Declarators:

### 3.1.1: Heelpoint:

**Object:** The **Heelpoint** keyword allows creating a heel angle, which is necessary whenever you need to refer to a given heel angle, i.e. for calculating downflooding angles, extrema, intersections… Constant angles are also sometimes necessary.

**Syntax 1:** ***Heelpoint*** *heelpoint("username", optional)*

This syntax allows creating easily any constant HeelPoint needed in the script.

- *heelpoint* is the internal name of the Heelpoint.

- *username* is the user name of the Heelpoint. Any string excluding quotation mark is valid.

- *optional* is an optional parameter depending on MNEMONIC.

The following examples are valid expressions:

```
Heelpoint theta0("Theta0", 0.0)
Heelpoint theta1("Theta1", heel)
Heelpoint theta2("Theta2", MIN(30.0, heel))
Heelpoint theta3("Theta3", {0.0})
Heelpoint theta4("Theta4", {heel})
Heelpoint theta5("Theta5", {109*k*x1*x2*((r*s)^0.5)})
Heelpoint theta6("Theta6", MIN(30, MAX(0,
{109*k*x1*x2*((r*s)^0.5)})))
```

**Syntax 2:** ***Heelpoint*** *heelpoint("username", MNEMONIC, optional1, optional2, optional3, optional4)*

- *heelpoint* is the internal name of the Heelpoint.

- *username* is the user name of the Heelpoint. Any string excluding quotation mark is valid.

- *MNEMONIC* is an STC mnemonic selecting how the Heelpoint is generated (see Mnemonics below).

- *optional1, optional2, optional3, optional4* are optional parameters depending on MNEMONIC.

**Mnemonics:**

- CONSTANT: This mnemonic allows creating a Heelpoint by specifying directly its heel angle. Since a direct and more general expression is possible (see syntax 1 above), this CONSTANT mnemonic is obsolete, being only supported for backward compatibility reasons. The following example is the only valid expression, where *ddd* represents any heel angle in degrees in a floating-point format:

```
Heelpoint theta0("Theta0", CONSTANT, ddd)
```

- MINHEEL: This mnemonic allows creating a Heelpoint representing the lowest heel of the calculated range. The following example is a valid expression:

```
Heelpoint minheel("Min Calculated Heel", MINHEEL)
```

- MAXHEEL: This mnemonic allows creating a Heelpoint representing the highest heel of the calculated range. The following example is a valid expression:

```
Heelpoint maxheel("Max Calculated Heel", MAXHEEL)
```

- HEELI: This mnemonic allows creating a Heelpoint containting the heel of $I^{th}$ calculated floatation. The *optional1* argument must therefore be a number or a Data defining index's value in the [0, imax(NWPLI)] range. The following example uses valid expressions:

```
Data Imax("Nb of calculated floatations", NWPLI)
Heelpoint minheel("Min Calculated Heel", HEELI, 0)
Heelpoint maxheel("Max Calculated Heel", HEELI, Imax)
```

- MAX: This mnemonic allows creating a Heelpoint corresponding to the max ordinate of the curve pointed by *optional1* (i.e. the heel angle coresponding to the maximum GZ value). The following example is a valid expression:

```
Heelpoint hgzmax("GZMax Heel", MAX, gzcurve)
```

- MIN: This mnemonic allows creating a Heelpoint corresponding to the min ordinate of the curve pointed by *optional1* (i.e. the heel angle coresponding to the minimum freeboard value). The following example is a valid expression:

```
Heelpoint hmin("Min Height Heel", MIN, hgtcurve)
```

- DYNCAPSZ: This mnemonic allows creating a Heelpoint corresponding to the dynamic capsizing angle calculated in the heel range according to the current ship balance angle pointed by *optional1*. The following example is a valid expression:

```
Heelpoint hdyncpsz("Dynamic Capsizing", DYNCAPSZ, heq)
```

- INTERSECTION: This mnemonic allows creating a Heelpoint corresponding to an intersection of the curves pointed by *optional1* and, if necessary, *optional2*. As there can be more than 1 intersection between 2 curves, the number of the intersection must be also specified in order to select it among the available ones. The intersection selection options are the followings:

- When *optional3* is one of the following mnemonics and *optional4* is not used, the intersection can be selected directly:

  - XMIN: Selects the intersection corresponding to the smallest heel.
  - ABSXMIN: Selects the intersection corresponding to the smallest absolute heel.
  - YMIN: Selects the intersection corresponding to the smallest ordinate.
  - ABSYMIN: Selects the intersection corresponding to the smallest absolute ordinate.
  - XMAX: Selects the intersection corresponding to the greatest heel.
  - ABSXMAX: Selects the intersection corresponding to the greatest absolute heel.
  - YMAX: Selects the intersection corresponding to the greatest ordinate.
  - ABSYMAX: Selects the intersection corresponding to the greatest absolute ordinate.

- When *optional3* is the name of an existing Heelpoint and *optional4* is not used, the retained intersection is the closest to *optional3* Heelpoint.

- When *optional3* is simply a positive integer, the result is the *optional3*$^{th}$ intersection of the curves starting from the smallest heel angle in the calculated range. This numbering type must be handled with care, as the results highly depend on calculated range and GZ curve configuration (by example, assuming ship balance heel to the first intersection of GZ and OX curves will provide an erroneous result when curves have a negative intersection).

- When *optional3* is the name of an existing Heelpoint and *optional4* is an integer value, it is considered as the intersection number referring to *optional3* origin, and this approach is the recommended one, being much more reliable than the previous:

  - *optional4*=1 selects the first intersection whose heel is **greater than** *optional3*
  - *optional4*=2 selects the second intersection whose heel is **greater than** *optional3*

  etc…

but also:

- *optional4*=-1 selects the first intersection whose heel is **smaller than** *optional3*
- *optional4*=-2 selects the second intersection whose heel is **smaller than** *optional3*

etc…

The following examples are valid expressions:

```
Heelpoint heq("Equilibrium", INTERSECTION, ox, gz, 1)// Not recommended
Heelpoint heq("Equilibrium", INTERSECTION, ox, gz, h0)// Recommended
Heelpoint hc("Static Capsizing", INTERSECTION, ox, gz, hgzmax, 1)// Recommended
Heelpoint hdfl("Downflooding", INTERSECTION, ox, openingfbd, heq, 1)// Recommended
```

- INTERSECTION+ (resp. INTERSECTION-): This mnemonic, available since Rev. 7.0, allows creating a Heelpoint corresponding to a **growing** (resp. decreasing) intersection of the curve pointed by *optional1* and, *optional2*, onwhich the growth test at intersection is done. As the second curve is the tested one, you therefore have to **avoid exchanging it with the first one**.

The interest of these mnemonics, compared with the previous one, is that they allow identifying growing (resp. decreasing) intersections directly, which is generally useful for distinguishing equilibrium and capsizing angles.

Like previously, as there can be more than 1 intersection between 2 curves, the number of the intersection must be also specified, in order to select it among the available ones. The intersection selection options are the followings:

- When *optional3* is one of the following mnemonics and *optional4* is not used, the intersection can be selected directly:

  - XMIN: Selects the intersection corresponding to the smallest heel.
  - ABSXMIN: Selects the intersection corresponding to the smallest absolute heel.
  - YMIN: Selects the intersection corresponding to the smallest ordinate.
  - ABSYMIN: Selects the intersection corresponding to the smallest absolute ordinate.
  - XMAX: Selects the intersection corresponding to the greatest heel.
  - ABSXMAX: Selects the intersection corresponding to the greatest absolute heel.
  - YMAX: Selects the intersection corresponding to the greatest ordinate.
  - ABSYMAX: Selects the intersection corresponding to the greatest absolute ordinate.

- When *optional3* is the name of an existing Heelpoint and *optional4* is not used, the retained intersection is the closest to *optional3* Heelpoint.

- When *optional3* is simply a positive integer, the result is the *optional3*th intersection of the curves starting from the smallest heel angle in the calculated range. This numbering type must be handled carefully, as the results highly depend on calculated range and GZ curve configuration (by example, assuming ship balance heel to the first intersection of GZ and OX curves will provide an erroneous result when curves have a negative intersection).

- When *optional3* is the name of an existing Heelpoint and *optional4* is an integer value, it is considered as the intersection number referring to *optional3* origin, and this approach is the recommended one, being much more reliable than the previous:

  - *optional4*=1 selects the first intersection whose heel is **greater than** *optional3*
  - *optional4*=2 selects the second intersection whose heel is **greater than** *optional3*

  etc…

  but also:

  - *optional4*=-1 selects the first intersection whose heel is **smaller than** *optional3*
  - *optional4*=-2 selects the second intersection whose heel is **smaller than** *optional3*

  etc…

The following example uses valid expressions:

```
Heelpoint heq("Equilibrium", INTERSECTION+, ox, gz, h0)
SetBalanceHeel(heq)
Heelpoint hc("Static Capsizing", INTERSECTION-, ox, gz, heq)//
```

### 3.1.2: Curve:

**Object:** The **Curve** keyword allows creating the necessary curves whenever you need to display, list, intersect or simply refer to given curves. Depending on the process, you may have to create auxiliary curves that won't have to be displayed on the stability diagram.

**Syntax 1:** `Curve curve("username", MNEMONIC, optional1, optional2)`

- `curve` is the internal name of the Curve.

- `username` is the user name of the Curve. Any string excluding quotation mark is valid.

- `MNEMONIC` is a STC mnemonic selecting the Curve type (see Mnemonics below).

- `optional1, optional2` are optional parameters depending on MNEMONIC.

**Mnemonics:**

- CONSTANT: This mnemonic allows creating a constant ordinate curve on the calculated heel range. The `optional1` parameter specifies the constant lever value. It can be the name of any previously created Lever or any constant value in the floating point format / current length units. The following examples are valid expressions:

```
Curve ox("Heel Axis", CONSTANT, 0.0)
Curve windcrv("Wind", CONSTANT, windlev)
```

- GZ: This mnemonic allows creating the transverse righting arm (GZ) curve on the calculated heel range. No `optional1` parameter is necessary. The following example is a valid expression:

```
Curve gzcurve("GZ Curve", GZ)
```

- GZL: This mnemonic allows creating the transverse-righting arm of liquids on the calculated heel range (i.e. the transverse-righting arm of liquids). No `optional` parameter is necessary. The following example is a valid expression:

```
Curve liquidgzcurve("Liquid's GZ Curve", GZL)
```

- DGZL: This mnemonic allows creating the transverse righting arm loss due to liquid's shift in the tanks on the calculated heel range (i.e. the difference GZ with liquids - GZ with frozen liquids). Its ordinates are negative as it represents a GZ loss. No `optional` parameter is necessary. The following example is a valid expression:

```
Curve dgzl("Tank's GZ Loss", DGZL)
```

- MS: This mnemonic allows creating the transverse residuary righting arm curve (MS curve) on the calculated heel range. No `optional` parameter is necessary. The following example is a valid expression:

```
Curve mscurve("MS", MS)
```

- GZAREA: This mnemonic allows integrating a given lever curve on the calculated heel range (i.e. creating the transverse righting arm area curve). The *optional1* parameter points on the lever curve to integrate (usually the GZ curve, which must therefore be created first) and the *optional2* parameter points on the heel angle (which must therefore also be created first) where the integrated area equals 0 (i.e. the equilibrium heel for GZ area curve). The following example is a valid expression:

```
Curve gzarea("GZ Area", GZAREA, gz, heq)
```

**Syntax 2:** ***Curve*** *curve("username", heel, {expression})*

This syntax allows creating a lever curve by specifying its algebraic expression directly. A heel HeelPoint defining the heel abscissa used in *expression* must, nevertheless, be created first, to allow its identification.

- *curve* is the internal name of the Curve.

- *username* is the user name of the Curve. Any string excluding quotation mark is valid.

- *heel* is the name of an existing HeelPoint representing the abscissa in the algebraic expression.

- *expression* is the algebraic expression of curve's ordinate, in which *heel* is used as abscissa. In addition to the classical +, -, *, /, ^ operators, the following transcendantal functions can also be used:

  - abs(val)
  - cos(val)
  - acs(val)
  - sin(val)
  - asn(val)
  - tan(val)
  - atn(val)
  - exp(val)
  - log(val)
  - l10(val)

Moreover, existing curves can also be used in *expression* provided that brackets delimiting the heel value follow their name.

The following examples are valid expressions:

```
Heelpoint heel("heel buffer", CONSTANT, 0.0)// Must be created first
Curve harm("Heeling Arm", heel, {armax*(cos(heel)^2)})// Defines harm curve as the product of armax lever and cos²(heel)
…
Curve gz("GZ", GZ)// Creates the gz curve…
Curve armdiff("Righting Arm - Heeling Arm", heel, {gz(heel)-harm(heel)})//… and creates the armdiff curve as the difference between gz and harm curves
```

**Syntax 3:** ***Curve*** *acurve("username", lcurve, heel, area)*

This syntax allows creating the *acurve* area curve associated to a given *lcurve* lever curve. A *heel* value and its corresponding *area* value must, nevertheless, also be specified, in order to define *acurve*'s integration constant. Usually, *heel* and *area* are locating *acurve*'s intersection with ox axis, so that *area* value is 0.0.

- *acurve* is the internal name of the Curve.

- *username* is the user name of the Curve. Any string excluding quotation mark is valid.

- *lcurve* is the name of the existing lever curve integrated for creating *acurve*.

- *heel* is the name, value or expression of the heel angle used for defining the integration constant.

- *area* is the name, value or expression of the area associated to *heel*, defining *acurve*'s integration constant.

The following examples are valid expressions:

```
Heelpoint thetac("Thetac", INTERSECTION, gz, harm, h0, 1)
Curve armdiff("GZ -Heeling Arm Lever", heel, {gz(heel)-harm(heel)})
Curve areadiff("GZ-Heeling Arm Area", armdiff, thetac, 0.0)// Creates
an arm difference area curve crossing point thetac , 0.0


Curve gz("GZ", GZ)// Creates the gz curve…
Heelpoint heq("Equilibrium", INTERSECTION, gz, ox, h0)
Curve gza("GZ Area", GZAREA, gz, heq)// … and creates its area curve…
Curve gza2("GZ Area again", gz, heq, 0.0)//… and recreates it by an
other way !
```

### 3.1.3: Area:

**Object:** The **Area** keyword allows creating the GZ areas needed for criterion calculation and diagram plotting by specifying the upper and lower lever curves (Curve names) and the heel limits (Heelpoint names).

**Syntax:** `Area area("username", curve1, curve2, heel1, heel2)`

- `area` is the internal name of the Lever.

- `username` is the user name of the Area. Any string excluding quotation mark is valid.

- `curve1` (resp. `curve2`) is the internal name of the lower (resp. upper) lever Curve limiting the Area.

- `heel1` (resp. `heel2`) is the internal name of the smallest (resp. greatest) Heelpoint limiting the Area.

**Mnemonics:**

No mnemonic is necessary, as one single syntax is available.

The following examples are valid expressions:

```
Area a030("[0, 30]", ox, gz, h0, h30)
Area a030("[eq., 30]Area", ox, gz, heq, MIN(hd, h30))
```

### 3.1.4: Lever:

**Object:** The **Lever** keyword allows creating the levers / righting arms needed by the criterion calculation. Levers are only supported for backward compatibility reasons, a more general **Data** variable being now available.

**Syntax:** Without mnemonic: ***Lever*** *lever("username", {expression})*
With mnemonic: ***Lever*** *lever("username"", MNEMONIC, heel)*

- *lever* is the internal name of the Lever.

- *username* is the user name of the Lever. Any string excluding quotation mark is valid.

- *MNEMONIC* is a mnemonic selecting the Lever type (see Mnemonics below).

- *{expression}* is any algebraic expression defining the lever value in current units.

- *heel* is the heel angle in degreees to which the lever pointed by MNEMONIC must be calculated.

**Mnemonics:**

- No mnemonic: When no mnemonic is specified, any simple or complex algebraic expression delimited between braces can specify lever's value. The following examples are valid expressions:

```
Lever critlever("Critical", {0.15})
Lever average("Mean Lever", {0.5*(lever1+lever2)})
```

- GZ, GMT, GMTC: These mnemonics allows calculating the corresponding parameter at a given heel, specified by *heel*. The following examples are valid expressions:

```
Lever gzx("GZ Max", GZ, gzmaxheel)// Heelpoint gzmaxheel must exist
Lever gm0("GMequ.", GMT, equ.)// Heelpoint equ. must exist
Lever gm0c("GMequ. cor.", GMTC, equilibrium)
```

### 3.1.5: InputLever:

**Object:** The **InputLever** keyword allows the user to enter the runtime definable Lever needed by criterion script (i.e. passengers or rafts lever…). The **InputLever** statement is only supported for backward compatibility reasons, as the new **InputData** statement is equivalent with a LEN mnemonic. The levers created by **InputLever** can be used like if they were created by **Lever**, except that the user can specify their value at runtime, updating the displayed data in real time. These user inputs are usually set at the beginning of the script, just after the Userprompt statements.

**Syntax:** ***InputLever*** *lever("input prompt", defaultvalue)*

- *lever* is the internal name of the user defined Lever.

- *input prompt* is the string to prompt in the user input area.

- *defaultvalue* is the initial value of *lever*, before the user has entered anything.

**Mnemonics:**

No mnemonic is needed, as this function is equivalent to using **InputData** with a LEN mnemonic.

The following examples are valid expressions:

```
InputLever psglev("Passengers Lever", 0.0)
InputLever trnlev("Turning Lever", 0.0
InputLever sd("Strain Ap.Y", 0.0)
InputLever sh("Strain Ap.Z", 0.0)
```

### 3.1.6: Data:

**Object:** The **Data** keyword allows creating all the variables needed in criterion scripts. Due to the growing complexity of stability criteria, the initially short list of variable types (Curve, Heelpoint, Lever, Area) had to be widely extended. The **Data** variable has therefore been added in order to allow representing any kind of variable. In order to allow unit conversion, data formatting…, a mnemonic is needed for defining data's dimension.

**Syntax 1:** `Data data("username", MNEMONIC, optional, optionalxylist…)`

- *data* is the internal name of the Data.

- *username* is the user name of the Data. Any string excluding quotation mark is valid.

- *MNEMONIC* is a mnemonic selecting the Data type (see Mnemonics below).

- *optional* is the data value or the name of the Heelpoint of the angle where the data pointed by *MNEMONIC* must be calculated.

- *optionalxylist…* is a list of x,y values in which the data value is interpolated according to *optional*.

**Mnemonics:**

Dimensional mnemonics (see list in 4.3): These mnemonics can be used in 2 distinct cases:

- When *optional* is an algebraic expression directly defining the data value, allowing to create the most current variables freely. The following examples are valid expressions:

```
Data pi("Pi", ABS, {3.14159})
Data Ls("Ls", LEN, {Lsmax-Lsmin})
Data N("N1+2*N2", INT, {N1+(2*N2)})
Data R("R", ABS, {1.0-(5000.0/ (Ls+(5.0*N)+15225) )})
...
```

- When *optional* is followed by a x, y interpolation list, *data*=y will be obtained by linear interpolation in the x, y list, according to x=*optional*. This syntax is specially designed for scripting standard data tables in which a value must be interpolated (no limit on the amount of x, y pairs). The following examples are valid expressions:

```
Data x1("X1", ABS, bd, 2.4, 1.0, 2.5, 0.98, 2.6, 0.96, 2.7, 0.95,
2.8, 0.93, 2.9, 0.91, 3.0, 0.9, 3.1, 0.88, 3.2, 0.86, 3.4, 0.82, 3.5,
0.8)
Data x2("X2", ABS, cb, 0.45, 0.75, 0.5, 0.82, 0.55, 0.89, 0.6,
0.95, 0.65, 0.97, 0.7, 1.0)
Data s("s", ABS, t, 6.0, 0.1, 7.0, 0.098, 8.0, 0.093, 12.0,
0.065, 14.0, 0.053, 16.0, 0.044, 18.0, 0.048, 20.0, 0.035)
```

Hydrostatic mnemonics (see list in 4.4): Using an hydrostatic mnemonic supposes that *optional* defines a heel angle (Heelpoint name or algebraic

expression between braces). The value of the hydrostatic parameter defined by *MNEMONIC* is then recovered in *data* for the specified heel angle. When no heel angle is specified, le smallest angle of the calculation range is taken by default, this omission being useful for certain parameters which are not heel dependent.

The following examples are valid expressions:

```
Data ax1("ax1", DAMX1)
Data lwl("Lwl", LWL, thetaeq)
Data ndmg("Number of Damaged Compartment(s)", NDCMP)
Data fdbeq("Residuary Freeboard at equilibrium", MINFBD, thetaeq)
```

**Syntax 2:** `Data data("username", MNEMONIC, value, heel)`

- *data* is the internal name of the Data.

- *username* is the user name of the Data. Any string excluding quotation mark is valid.

- *MNEMONIC* is a mnemonic selecting the Data type (see Mnemonics below).

- *value* is the value on which *data* depends (according to *heel* angle).

- *heel* is the name, value or expression of the heel angle on which data must be evaluated, according to *value*.

**Mnemonics:**

- DWX: This mnemonic allows calculating silhouette's draft at *value*% between fore and aft silhouette's intersections with floatation at *heel* °. The following example is a valid expression, calculating silhouette's draft at 40% of Lwl, for *heq* heel:

```
Data draft_at_40pct("Draft @ 40% of Lwl", DWX, {40.0}, heq)
```

**Syntax 3:** `Data data("username", lcurve, heel)`

- *data* is the internal name of the Data.

- *username* is the user name of the Data. Any string excluding quotation mark is valid.

- *lcurve* is the name of an existing curve whose value will be evaluated at *heel* angle.

- *heel* is the name, value or expression of the heel angle on which *lcurve* will be evaluated for setting *data*'s value.

**Mnemonics:**

No mnemonic is needed as *lcurve* automatically defines *data*'s dimension.

The following examples are valid expressions:

```
Data gz30("GZ 30°", gz, 30.0)
Data gzmax("Max GZ", gz, gzmaxheel)
Data fbeq("Equilibrium Fbd ", fbdcurve, heq)
Data fsm("FSM", fsm, {heel*0.5*lambda*(3.0-lambda)})
```

### 3.1.7: InputData:

**Object:** The **InputData** keyword allows the user to enter the runtime definable data needed by criterion script (i.e. heel range to display, number of passengers or rafts…). Each **InputData** command adds one input field in the stability results dialog box. The data created by **InputData** can be used like if they were created by **Data**, except that the user can specify their value at run time, updating the displayed data in real time. These user inputs are usually set at the beginning of the script, just after the Userprompt statements.

**Syntax:** ***InputData*** *data("input prompt", MNEMONIC, defaultvalue)*

- *data* is the internal name of the user defined Data.

- *input prompt* is the string to prompt in the user input area.

- *MNEMONIC* is a dimension mnemonic selecting the Data type (see list below in 4.3).

- *defaultvalue* is the initial value of *data*, before the user has entered anything.

**Mnemonics:**

See list in 3.3.

The following examples are valid expressions:

```
InputData dhmin("Min Displayed Heel", Heel°, 0.0)
InputData dhmax("Max Displayed Heel", Heel°, 70.0)
InputData subl("Subdivision Length Ls (0 for Auto)", LEN, 0.0)
InputData aftorg("Subdivision Length Origin (0 for Auto)", LEN, 0.0)
InputData bmax("B (0 for Bwl)", LEN, 0.0) //
InputData N1("Number of Salvable Passengers (0 for cargo)", INT, 0)
InputData N2("Number of Extra Passengers (0 for cargo)", INT, 0)
InputData mrft("Rafts Mmt", MMT, 0) //
```

## *3.2* STC Commands:

### 3.2.1: Abort:

**Object:** The **Abort** keyword allows aborting script execution and displaying a warning message when abnormal results are found. This statement is usually conditional, depending on an If(…) test.

**Syntax:** ***Abort****("abortmessage")*

- *abortmessage* is the string defining the message that will be displayed to the user in a warning dialog box.

The following example is a valid expression:

```
If(a, <, 0.001)
 Abort("No Silhouette Descriptor for Windage Calculation")
EndIf
```

### 3.2.2: AddCSVLineItem:

**Object:** The **AddCSVLineItem** command allows feeding current table's current line by adding new items (cells / columns) in it, one by one.

Current line's first **AddCSVLineItem** statement therefore allows simultaneously defining its first cell content and its associated first column title. Similarly, the following **AddCSVLineItem** statements allow defining following cell's contents and column titles.

In multi-line tables, the last column title overwrites the previous one.

**Syntax 1:** ***AddCSVLineItem****("text")*

This syntax allows inserting remote text(s) / title(s) without changing column titles.

- *text* is the text to insert in the table. Any string excluding quotation mark and semicolon is valid.

The following example uses a valid expression:

```
StartCSVTable()// creates a new table and its first CSV line
  AddCSVLineItem("This is an example")//Adds a text in current line's
first column
EndCSVTable()// closes and validates the current table
```

**Syntax 2:** ***AddCSVLineItem****(MNEMONIC)*

This syntax allows inserting the current name defined by *MNEMONIC* without changing column titles.

- *MNEMONIC* must be selected in this range:

  - *CURNTNAME*: Outputs current model's name.
  - *CURNTLOADING*: Outputs current loading's name or 'Current Loading' if undefined.
  - *CURNTDAMAGE*: Outputs current damage's name or 'Current Damage' if undefined .

The following example uses valid expressions:

```
StartCSVTable()// creates a new table and its first CSV line
AddCSVLineItem(CURNTLOADING)
AddCSVLineItem(CURNTDAMAGE)
AddCSVLineItem(CURNTNAME)
EndCSVTable()// closes and validates the current table
```

**Syntax 3:** ***AddCSVLineItem****("column title", MNEMONIC, optional)*

This syntax allows inserting a new item / cell in the current CSV line by specifying its column title, hydrostatic calculation mnemonic and optional heel angle (default value is 0°).

- *column title* is the title of current item / cell's column (in multi-line tables, the last title of the column overwrites its previous one). Any string excluding quotation mark and semicolon is valid.

- *MNEMONIC* is an hydrostatic calculation mnemonic selecting which calculated data is exported in current cell (see "Hydrostatic Calculation Mnemonics" table below).

- *optional* is the optional heelpoint to which the data pointed by *MNEMONIC* must be calculated / interpolated. A 0° heel is the default value when *optional* is omitted.

The following multi-table example uses various valid expressions:

```
StartCSVTable()// creates a new table and its first CSV line
  AddCSVLineItem("Displ.", DSPL)//Adds (0°) displacement / «Displ.»
title in 1st cell
  AddCSVLineItem("LCB", LCB, heq)//Adds LCB @ heq heel / «LCB» title in
2nd cell
 AddCSVLineItem("VCB", VCB, heq)//Adds VCB @ heq heel / «VCB» title in
3rd cell
  AddCSVLineItem("TCB", TCB, heq)//Adds TCB @ heq heel / «VCB» title in
4th cell
  NewCSVLine()// validates CSV line with its 4 cells and creates a new
one
EndCSVTable()// closes and validates the current table

StartCSVTable()

  Data wplmax("Nb of calculated floatations", NWPLI)// Max nb of
calculated floatations
  Data wpli("Floatation index", INT, 0)//Current floatation index

  For(wpli, 0, <, wplmax)

    Heelpoint heeli("ith heel", HEELI, wpli)

    If(heeli, >=, dhmin)
      If(heeli, <=, dhmax)
        AddCSVLineItem("Heel°", Heel°, heeli)
        AddCSVLineItem("Lwl", LWL, heeli)
        AddCSVLineItem("Bwl", BWL, heeli)
        AddCSVLineItem("TOA", TOA, heeli)
        AddCSVLineItem("WS", WS, heeli)
        NewCSVLine()
      EndIf
    EndIf

  Next(1)

EndCSVTable()
```

**Syntax 4:** ***AddCSVLineItem****("column title", curve, heel)*

This syntax allows inserting a new item in current CSV line by specifying its column title, a curve and an associated heel angle.

- *column title* is the title of current cell's column (in multi-line tables, the last title of the column overwrites its previous one). Any string excluding quotation mark and semicolon is valid.

- *curve* is an existing curve name selecting the *curve* used for calculating cell's value at *heel* angle.

- *heel* is the heel angle or heelpoint to which *curve* must be evaluated for getting cell's value.

The following example uses a valid expression:

```
Curve gz("GZ", GZ)
Curve ox("Heel Axis", CONSTANT, 0.0)
Heelpoint heq("Ship Balance", INTERSECTION, gz, ox, h0)//Finds ship
balance
Curve fbmin("Min Freeboard", MINFBD)// Creates a freeboard(heel) curve

StartCSVTable()// creates a new table and its first CSV line
  AddCSVLineItem("Freeboard @ Equilibrium", fbmin, heq)//Adds Fbd @ heq
in cell
  NewCSVLine()// validates CSV line and creates a new one
EndCSVTable()// closes and validates the current table
```

**Syntax 5:** ***AddCSVLineItem****("column title", {expression}, MNEMONIC)*

This syntax allows inserting a new item in current CSV line, by specifying its column title, algebraic expression and dimensional mnemonic (for current units conversion).

- *column title* is the title of current cell's column (in multi-line tables, the last title of the column overwrites its previous one). Any string excluding quotation mark and semicolon is valid.

- *expression* is the algebraic expression of cell's data. Any existing variable can be used in this expression and, in addition to the brackets and classical +, -, *, /, ^ operators, the following transcendantal functions can also be used:

  - abs(…)
  - cos(…)
  - acs(…)
  - sin(…)
  - asn(…)
  - tan(…)
  - atn(…)
  - exp(…)
  - log(…)
  - l10(…)

- *MNEMONIC* is a dimensional mnemonic selecting expression's dimension to allow its conversion in current units (see "Dimensional Mnemonics" table below).

The following example uses a valid expression:

```
Data dft("TOA", TOA)

StartCSVTable()// creates a new table and its first CSV line
  AddCSVLineItem("Corrected Half Draft", {abs(0.1+(0.95*(0.5*dft)))},
LEN)
  NewCSVLine()// validates second CSV line with its single text and
creates a new one
EndCSVTable()// closes and validates the current table
```

As a conclusion, here is a complete example and its EXCEL result after CSV data Export:

```
Data dft("TOA", TOA)

StartCSVTable()
  AddCSVLineItem("Voici un exemple de texte libre")
  NewCSVLine()
EndCSVTable()

StartCSVTable()
  AddCSVLineItem("Equilibrium°", heq, ANG)
  AddCSVLineItem("Displ.", DSPL)
  AddCSVLineItem("LCB", LCB, heq)
  AddCSVLineItem("VCB", VCB, heq)
  AddCSVLineItem("TCB", TCB, heq)
  AddCSVLineItem("Half Draft", {0.5*dft}, LEN)
  NewCSVLine()
  NewCSVLine()
EndCSVTable()

StartCSVTable()

  Data wplmax("Nb of calculated floatations", NWPLI)// Max number of
calculated floatations
  Data wpli("Floatation index", INT, 0)//Current floatation index

  For(wpli, 0, <, wplmax)

    Heelpoint heeli("ith heel", HEELI, wpli)

    If(heeli, >=, dhmin)
      If(heeli, <=, dhmax)
        AddCSVLineItem("Heel°", Heel°, heeli)
        AddCSVLineItem("Lwl", LWL, heeli)
        AddCSVLineItem("Bwl", BWL, heeli)
        AddCSVLineItem("TOA", TOA, heeli)
        AddCSVLineItem("WS", WS, heeli)
        AddCSVLineItem("GZ", gz, heeli)
        NewCSVLine()
      EndIf
    EndIf

  Next(1)

EndCSVTable()
```

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | | | | | | |
| 2 | | | | | | |
| 3 | Voici un exemple de texte libre | | | | | |
| 4 | | | | | | |
| 5 | Equilibrium | Displ. | LCB | VCB | TCB | Half Draft |
| 6 | 0.003 | 246.299 | 11.1016 | 1.9589 | 0.0001 | 1.7862 |
| 7 | | | | | | |
| 8 | | | | | | |
| 9 | Heel° | Lwl | Bwl | TOA | WS | GZ |
| 10 | 0.000 | 23.5298 | 5.9249 | 3.8558 | 211.60 | 0.0000 |
| 11 | 0.003 | 23.5298 | 5.9249 | 3.8558 | 211.60 | 0.0000 |
| 12 | 2.500 | 23.5385 | 5.9193 | 3.8515 | 211.64 | 0.0359 |
| 13 | 5.000 | 23.5433 | 5.9029 | 3.8383 | 211.58 | 0.0720 |
| 14 | 7.500 | 23.5462 | 5.8759 | 3.8160 | 211.56 | 0.1085 |
| 15 | 10.000 | 23.5478 | 5.9200 | 3.7844 | 211.50 | 0.1458 |
| 16 | 12.500 | 23.5485 | 6.0797 | 3.7433 | 211.47 | 0.1840 |
| 17 | 15.000 | 23.5483 | 7.1134 | 3.6929 | 211.88 | 0.2230 |
| 18 | 17.500 | 23.5480 | 7.1252 | 3.6353 | 213.14 | 0.2609 |
| 19 | 20.000 | 23.5486 | 7.1570 | 3.5723 | 214.82 | 0.2968 |
| 20 | 22.500 | 23.5508 | 6.3283 | 3.5051 | 216.71 | 0.3292 |
| 21 | 25.000 | 23.5549 | 6.3786 | 3.4334 | 218.56 | 0.3568 |
| 22 | 27.500 | 23.5608 | 6.4176 | 3.3605 | 220.28 | 0.3803 |
| 23 | 30.000 | 23.5678 | 6.4492 | 3.2817 | 221.93 | 0.4008 |
| 24 | 32.500 | 23.5757 | 6.4995 | 3.3025 | 223.40 | 0.4189 |
| 25 | 35.000 | 23.5844 | 6.4853 | 3.3492 | 224.67 | 0.4351 |
| 26 | 37.500 | 23.5937 | 5.5329 | 3.3883 | 225.81 | 0.4490 |
| 27 | 40.000 | 23.6037 | 5.2681 | 3.4191 | 227.07 | 0.4610 |
| 28 | 42.500 | 23.6144 | 5.0384 | 3.4417 | 228.92 | 0.4717 |
| 29 | 45.000 | 23.6289 | 4.8296 | 3.4911 | 230.82 | 0.4819 |
| 30 | 47.500 | 23.6478 | 4.5886 | 3.5398 | 233.35 | 0.4908 |
| 31 | 50.000 | 23.6703 | 4.2061 | 3.5826 | 236.44 | 0.4951 |

### 3.2.3: AddList:

**Object:** The **AddList** keyword allows adding extra data list(s) to the default data list when it is not sufficient. The additional data to list must be specified after the AddList command, exactly as they are specified for the defaault list (see example below).

**Syntax:** ***AddList()***

The following example allows listing the multibody data according to their number:

```
Data nbody("nbody", NBODY, 0.0)

if(nbody, >, 0)
  AddList()
  ListTab(Heel°, "Heel:", 26, 26)
  ListTab(H0, "Heel0:", 36, 36)
  ListTab(T0, "Trim0:", 46, 46)
  ListTab(TM0, "MP Height0:", 60, 58)
  ListTab(TH0, "Theta0:", 71, 70)
  ListTab(FR0, "1st React.:", 84, 82)
  ListTab(LR0, "2nd React.:", 97, 94)
EndIf

if(nbody, >, 1)
  AddList()
  ListTab(Heel°, "Heel:", 26, 26)
  ListTab(H1, "Heel1:", 36, 36)
  ListTab(T1, "Trim1:", 46, 46)
  ListTab(TM1, "MP Height1:", 60, 58)
  ListTab(TH1, "Theta1:", 71, 70)
  ListTab(FR1, "1st React.:", 84, 82)
  ListTab(LR1, "2nd React.:", 97, 94)
EndIf

if(nbody, >, 2)
  AddList()
  ListTab(Heel°, "Heel:", 26, 26)
  ListTab(H2, "Heel2:", 36, 36)
  ListTab(T2, "Trim2:", 46, 46)
  ListTab(TM2, "MP Height2:", 60, 58)
  ListTab(TH2, "Theta2:", 71, 70)
  ListTab(FR2, "1st React.:", 84, 82)
  ListTab(LR2, "2nd React.:", 97, 94)
EndIf

if(nbody, >, 3)
  AddList()
  ListTab(Heel°, "Heel:", 26, 26)
  ListTab(H3, "Heel3:", 36, 36)
  ListTab(T3, "Trim3:", 46, 46)
  ListTab(TM3, "MP Height3:", 60, 58)
  ListTab(TH3, "Theta3:", 71, 70)
  ListTab(FR3, "1st React.:", 84, 82)
  ListTab(LR3, "2nd React.:", 97, 94)
EndIf
```

### 3.2.4: AreaRange:

**Object:** The **AreaRange** keyword allows controlling the GZ area range used for scaling GZ Area curve on stability diagram. The “wildcard” AUTO allows automatic scaling on current curve. Numeric values are interpreted in script’s area unit.

**Syntax:** ***Area****(areamin, areamax)*

- *areamin* defines the minimum area shown on stability diagram. It can be a numeric value, an Area name or an algebraic expression quoted between braces. The wildcard AUTO can also be used for automatic setting.

- *areamax* defines the minimum area shown on stability diagram. It can be a numeric value, an Area name or an algebraic expression quoted between braces. The wildcard AUTO can also be used for automatic setting.

The following examples are valid expressions:

```
AreaRange(-0.5, 0.5)
AreaRange(amin, amax)
AreaRange(amin, AUTO)
LeverRange(AUTO, AUTO)
```

### 3.2.5: CalculateWindageData:

**Object:** The **CalculateWindageData** keyword allows integrating ship's silhouette data at a given heel angle (usually 0° or ship's equilibrium) with the wind previously defined with the **WindGradient** statement, which must therefore necessarily precede it. This calculation can only be done if one or more silhouette descriptors (i.e. closed lines describing ship's silhouette in the appropriate layer) are present in the calculated model. Windage calculations can be easily avoided when no silhouette descriptor is available (i.e. integrated windage area obtained with mnemonic AWDequals 0) , as show in the example:

**Syntax:** ***CalculateWindageData(****heelangle)*

- *heelangle* defines the heel angle towhich ship's silhouette must be integrated. The results of this calculation can be displayed with "Silhouette Analysis" outputs.

The following example is a valid expression:

```
WindGradient(CONSTANT, OW, PNM², 1, 316, 2, 386, 3, 429, 4, 460, 5, 485, 6, 504)
CalculateWindageData(heq)

Data a("A", AWD)
If(a, <, 0.001) Abort("No Silhouette Descriptor for Windage Calculation")
```

### 3.2.6: DefineCargoHeel:

**Object:** Cargo's free surface may differ from waterplane's free surface, depending on cargo's viscosity. The **DefineCargoHeel** keyword allows defining how cargo's free surface varies according to waterplane's heel and which fictitious density must be used for calculating such cargo's FSM. This feature is especially needed for dredge stability calculations.

This command being processed before stability calculations, **DefineCargoHeel** must necessarily be used at the beginning of the script, before any operation on stability data, which are unknown at this preprocess step. It is therfore a good habit to use **DefineCargoHeel** in script's header.

For further details, See MAAT Hydro's FAQ >> Calculation >> How to Calculte Dredge's Stability ?

**Syntax:** ***DefineCargoHeel****(cargoheelcurve, fsmdensity)*

- *cargoheelcurve* is the curve defining cargo's heel according to heel (must be previously created).

- *fsmdensity* is the density to be used for calculating cargo's FSM.

The following example is valid when located in script's header:

```
Data lambda("Lambda", CARGODENS) // Creates a Cargo Density data…
Data lambdaf("Lambdaf", DNS, {0.5*lambda*(3.0-lambda)})//…and
associated FSM density
Heelpoint heel("heel buffer", CONSTANT, 0.0)//  Creates a heel abscissa
buffer
Curve cargoheel("Cargo Heel", heel, {0.5*(3-lambda)*heel})//Creates the
cargo heel curve according to heel
DefineCargoHeel(cargoheel, lambdaf)// Defines cargo heel function
according to heel
```

### 3.2.7: DefineSubmersionRange:

**Object:** Water on Deck's height is linearly interpolated according to the residuary freeboard at equilibrium between the damage limits, and then weighted by a factor depending on swell's service height (see below).

The '***DefineSubmersionRange***' statement allows defining this interpolation range and the corresponding Water heights. This feature is especially necessary for stability calculations with Water on Deck.

As this command is processed before stability calculations, **DefineSubmersionRange** must necessarily be used at the beginning of the script, before any operation on stability data, which are unknown at this preprocess step. It is therfore a good habit to use **DefineSubmersionRange** only in script's header.

For further details on Water on Deck / RoRo stability, see MAAT Hydro's FAQ >> Calculation >> How to Calculate RoRo's damage stability according to Stockholm regulations

**Syntax:** ***DefineSubmersionRange****(freebd1, WoDheight1, freebd2, WoDheight2)*

- *freebd1*: First residuary freeboard.

- *WoDheight1:* Associated Water's height on deck.

- *freebd2*: Second residuary freeboard.

- *WoDheight2:* Associated Water's height on deck.

The following example is valid when located in script's header: It means that Water on Deck's height will vary linearly between 0.5m, corresponding to residuary freeboards smaller than 0.3 m and 0 m, corresponding to residuary freeboards greater than 2 m:

```
// Stockholm Data ranges must be defined first to make the Criterium usable
for WoD/RoRo Stability Calculations:
DefineSubmersionRange(0.3, 0.5, 2.0, 0.0)//  Defines Min and Max Submersion
range limits and the associated added water heights
DefineSwellRange(1.5, 4.0)//  Defines Min and Max Swell range limits
// End ofStocholm Data
```

### 3.2.8: DefineSwellRange:

**Object:** Water on Deck's height is linearly interpolated according to the residuary freeboard at equilibrium in the damage range (see above), and then weighted by a factor depending on swell's average height in service area.

The '***DefineSwellRange***' statement allows defining the swell range in which Water on Deck's height will be weighted from 0% to 100%, according to swell's average height in service area. This feature is especially necessary for stability calculations with Water on Deck.

As this command is processed before stability calculations, **DefineSwellRange** must necessarily be used at the beginning of the script, before any any operation on stability data, which are unknown at this preprocess step. It is therfore a good habit to use **DefineSwellRange** only in script's header.

For further details on Water on Deck / RoRo stability, see MAAT Hydro's FAQ >> Calculation >> How to Calculate RoRo's damage stability according to Stockholm regulations

**Syntax:** ***DefineSwellRange****(heightmin, heightmax)*

- *heightmin*: Min swell's height, corresponding to 0% of Water on Deck's height.
- *heightmax*: Max swell's height, corresponding to 100% of Water on Deck's height.

The following example is valid when located in script's header: It means that Water on Deck's height will vary linearly between 0.5m, corresponding to residuary freeboards smaller than 0.3 m and 0 m, corresponding to residuary freeboards greater than 2 m:

```
// Stockholm Data ranges must be defined first to make the Criterium usable
for WoD/RoRo Stability Calculations:
DefineSubmersionRange(0.3, 0.5, 2.0, 0.0)//  Defines Min and Max Submersion
range limits and the associated added water heights
DefineSwellRange(1.5, 4.0)//  Defines Min and Max Swell range limits
// End ofStocholm Data
```

### 3.2.9: Else:

**Object:** The **Else** keyword allows defining two complementary conditional processes. When the condition defined by **If**(…) is satisfied, the process following this statement is executed till reaching the associated **EndIf** or **Else** statement. When the condition is not satisfied, this first process is ignored, an **Else** statement allowing to execute the alternative process following it, till reaching the associated **EndIf**. The **If** / **Else** / **EndIf** statements are well know of programmers and don't need special explainations. The **If / Endif** statements are necessary for setting the optional **Else**.

**Syntax:** ***Else***

**Else** doesn't need any argument, but depends on the corresponding **If**(…) / **Endif** statements.

The following example is a valid expression:

```
      if(bmax, =, 0)
        // This process is only executed when bmax is equal to 0 (bmax is
user defined)
      Else
        // This alternative process is only executed when bmax is not equal
to 0
      EndIf
```

### 3.2.10: EndCSVTable:

**Object:** As this command closes the table definition started with **StartCSVTable** (see above for further details), both commands must always match together.

**Syntax:** ***EndCSVTable****()*

**EndCSVTable** doesn't need any argument but must always match with a previously associated **StartCSVTable** for validating the table lines and cells defined between this delimitor pair.

The following example is a valid expression:

```
StartCSVTable()// creates a new table and its first CSV line
  AddCSVLineItem("This is an example ")// inserts text in the table
  NewCSVLine()// skips a line
EndCSVTable()// closes and validates the current table
```

### 3.2.11: EndIf:

**Object:** The **EndIf** keyword closes the conditional process opened by **If**. The **If** / **Else** / **EndIf** statements are well know of programmers and don't need special expainations. Each **If** statement needs to be closed by an **EnIf** statement, the **Else** statement being optional.

**Syntax:** ***EndIf***

**EndIf** doesn't need any argument, but must match with a previous **If**(…) statement.

The following examples are valid expressions:

```
if (trnlev, >, 0)
  // Only executed when trnlev is strictly greater than 0
EndIf

if (intcounter, !=, 0)
  // Only executed when intcounter is different of 0
EndIf

if (bmax, =, 0)
  // Only executed when bmax is equal to 0 (bmax is user defined)
Else
  // Only executed when bmax is not equal to 0
EndIf
```

### 3.2.12: For:

**Object:** The **For** keyword allows repeating a process as long as the execution condition is satisfied, provided that it is associated to a corresponding **Next** Statement that increments the loop counter. These **For** / **Next** statements being well known of programmers, don't seem to need special explainations here.

**Syntax:** ***For(****countername, initialvalue, MNEMONIC, finalvalue)*

- *countername* is the name of data used as loop counter (must be a **Data** name, usually of INT type).

- *initialvalue* is the initial counter value.

- *MNEMONIC* defines the test operation (see list of available mnemonics below).

- *finalvalue* is the value compared with *counter,* according to *MNEMONIC,* in order to decide if loop must be reexecuted.

**Mnemonics:**

- **>**: Executes the lines between **For** and **Next** statements as long as *countername* is strictly greater than *finalvalue*.
- **>=**: Executes the lines between **For** and **Next** statements as long as *countername* is equal or greater than *finalvalue*.
- **<**: Executes the lines between **For** and **Next** statements as long as *countername* is strictly smaller than *finalvalue*.
- **<=**: Executes the lines between **For** and **Next** statements as long as *countername* is equal or smaller than *finalvalue*.
- **=**: Executes the lines between **For** and **Next** statements as long as *countername* is equal to *finalvalue*. This test is not reliable when applied to non integer counters.
- **!=**: Executes the lines between **For** and **Next** statements as long as *countername* is different of *finalvalue*. This test is not reliable when applied to non integer counters.

The following examples, shared with the **Next** statement, are valid expressions:

```
Data cpt("1st loop counter", INT, {0})//

For(cpt, 0, >=, -5)
  // process to repeat as long as cpt>=-5
Next(-1)// decrements cpt

Data i("2nd loop counter", INT, {0})//
Data j("3rd loop counter", INT, {0})//

For(i, 0, <, 3)

  // process to repeat as long as i<3

  For(j, 0, >=, -5)
    // process to repeat as long as i<3 and j>=-5
```

```
  Next(-1) // decrements j

Next(1)// increments i

If(n, =, 1)
  Data imax("imax", INT, {1})//
Else
  If(n, =, 2)
    Data imax("imax", INT, {3})//
  Else
    Data imax("imax", INT, {4})//
  EndIf
EndIf

For(i, 0, <, imax)
  // process to repeat as long as I<imax
Next(1)//increments i
```

### 3.2.13: HeelRange:

**Object:** The **HeelRange** keyword allows controlling the heel range shown on stability diagram. This heel range can, of course, not exceed the calculated range. The “wildcard” AUTO allows automatic scaling on current curve.

**Syntax:** ***HeelRange(****heelmin, heelmax)*

- *heelmin* defines the minimum heel angle shown on stability diagram. It can be a numeric value, a Heelpoint name or an algebraic expression quoted between braces. The wildcard AUTO can also be used for automatic setting.

- *heelmax* defines the maximum heel angle shown on stability diagram. It can be a numeric value, a Heelpoint name or an algebraic expression quoted between braces. The wildcard AUTO can also be used for automatic setting.

The following examples are valid expressions:

```
HeelRange(0, 70)
HeelRange(hmin, hmax)
HeelRange(hmin, AUTO)
HeelRange(AUTO, AUTO)
```

### 3.2.14: If:

**Object:** The **If** keyword is associated to EndIf and Else Statements. These three classical statements well know of programmers, allow conditional processes. Each **If** statement needs to be closed by an **EnIf** statement, the Else statement being optional.

**Syntax:** *If (variable, MNEMONIC, threshold)*

- *variable* is the name of variable to test.

- *MNEMONIC* defines the test operation (see list of available mnemonics below).

- *threshold* is the name or the value of the critical test value.

**Mnemonics:**

- **>**: Executes the lines following **If** statement when *variable* is strictly greater than *threshold*
- **>=**: Executes the lines following **If** statement when *variable* is equal or greater than *threshold*
- **<**: Executes the lines following **If** statement when *variable* is strictly smaller than *threshold*
- **<=**: Executes the lines following **If** statement when *variable* is equal or smaller than *threshold*
- **=**: Executes the lines following **If** statement when *variable* equal to *threshold*. For obvious reasons, this test is only reliable on INT *variable* and *threshold*.
- **!=**: Executes the lines following **If** statement when *variable* is different of *threshold*. For obvious reasons, this test is only reliable on INT *variable* and *threshold*.

The following examples are valid expressions:

```
if (trnlev, >, 0)
  // Only executed when trnlev is strictly greater than 0
EndIf

if (intcounter, !=, 0)
  // Only executed when intcounter is different of 0
EndIf

if (bmax, =, 0)
  // Only executed when bmax is equal to 0 (bmax is user defined)
Else
  // Only executed when bmax is not equal to 0
EndIf
```

### 3.2.15 LeverRange:

**Object:** The **LeverRange** keyword allows controlling the lever range used for scaling GZ and lever curves on stability diagram. The “wildcard” AUTO allows automatic scaling on current curve. Numeric values are interpreted in script’s length unit.

**Syntax:** ***LeverRange(****levermin, levermax)*

- *levermin* defines the minimum lever shown on stability diagram. It can be a numeric value, a Lever name or an algebraic expression quoted between braces. The wildcard AUTO can also be used for automatic setting.

- *levermax* defines the maximum lever shown on stability diagram. It can be a numeric value, a Lever name or an algebraic expression quoted between braces. The wildcard AUTO can also be used for automatic setting.

The following examples are valid expressions:

```
LeverRange(-0.5, 0.5)
LeverRange(lmin, lmax)
LeverRange(lmin, AUTO)
LeverRange(AUTO, AUTO)
```

### 3.2.16: List:

**Object:** The **List** keyword allows inserting additional heel angles, which are not necessarily round angles, to the initially calculated heel angle list. These additional heel angles (i.e. ship's equilibrium, capsizing…) will then insert new lines in the data list following the stability diagram.

**Syntax:** ***List****(heelangle)*

- *heelangle* is the internal name of Heelpoint to insert in the data list.

The following examples are valid expressions:

```
List(equilibrium)
List(dyncapsizing)
List(downflooding)
```

### 3.2.17: ListTab:

**Object:** The **ListTab** keyword allows defining the hydrostatic data to print and their position in the list following the stability diagram.

**Syntax 1:** ***ListTab****(DATAMNEMONIC, "columntitle", titletab, valuetab)*

- *DATAMNEMONIC* defines the hydrostatic data to print in the lising (see list of available hydrostatic mnemonics below, in section 4.4).

- *"columntitle"* is the title of the corresponding colums of the list. It may therfore be different of the default data name.

- *titletab* is the printing column (counted in characters, 1 being the leftmost) of the column title *columntitle*.

- *valuetab* is the printing column (counted in characters, 1 being the leftmost) of data's value. *titletab* and *valuetab* may be different for layout reasons, as their length may differ.

**Mnemonics:**

Any hydrostatic mnemonic listed in section 4.4 is valid for *DATAMNEMONIC*.

The following examples are valid expressions (corresponding to the layout of most of the default criteria):

```
ListTab(Heel°, "Heel:", 26, 26)
ListTab(TA, "AP Draft:", 38, 36)
ListTab(TF, "FP Draft:", 48, 46)
ListTab(TCB, "TCB:", 53, 54)
ListTab(VCB, "VCB:", 60, 62)
ListTab(LCB, "LCB:", 69, 70)
ListTab(KN, "KN:", 77, 78)
ListTab(MS, "MS:", 85, 86)
ListTab(GZ, "GZ:", 93, 94)
ListTab(GZArea, "Area:", 102, 102)
```

**Syntax 2:** ***ListTab****(curve, "columntitle", titletab, valuetab)*

- *curve* defines the curve whose ordinates must be printed in the lising.
- *"columntitle"* is the title of the corresponding colums of the list. It may therfore be different of the default data name.

- *titletab* is the printing column (counted in characters, 1 being the leftmost) of the column title *columntitle*.

- *valuetab* is the printing column (counted in characters, 1 being the leftmost) of data's value. *titletab* and *valuetab* may be different for layout reasons, as their length may differ.

The following examples are valid expressions:

```
Curve gz("GZ", GZ)// Creates the gz curve…
ListTab(gz, "GZ:", 93, 94)
Curve gza("GZ Area", GZAREA, gz, heq)// … and creates its area curve
ListTab(gza, "Area:", 102, 102)
```

### 3.2.18: Maat Stability Criterion:

**Object:** The **MaatStabilityCriterion** keyword (blanks are ignored) allows identifying STC scripts and naming the criterion. This statement must therefore be the first line of a STC script.

**Syntax:** ***Maat Stability Criterion:*** *"criterion name"*

- *criterion name* is the string defining the criterion name to prompt in the user input area.

The following examples is a valid expression:

```
Maat Stability Criterion: "IMO A749 (18) - Round Bilge -"
```

### 3.2.19: NewCSVLine:

**Object:** The **NewCSVLine** command allows validating the current CSV line and creating a new empty one, incrementing table's line number like a "Carriage Return / Line Feed" should do on a typewriter.

Such CSV lines must be fed item by item (cell by cell / column by column) with **AddCSVLineItem** statements (see below).

**Syntax:** ***NewCSVLine****()*

**NewCSVLine** doesn't need any argument.

The following example uses valid expressions:

```
StartCSVTable()// creates a new table and its first CSV line
  NewCSVLine()// validates first CSV line as blank and creates a
new one
  AddCSVLineItem("This is an example")//Adds a text in current
line's first column
  NewCSVLine()// validates second CSV line with its single text
and creates a new one
EndCSVTable()// closes and validates the current table
```

### 3.2.20: Next:

**Object:** The **Next** keyword marks the end of the loop previously opened by a **For** statement and increments / decrements the associated loop counter before repeating the delimited statements as long as the execution condition is satisfied. A special attention must be paid to this execution condition in order to avoid infinite loops. These **For** / **Next** statements being well known of programmers, don't seem to need special explainations here. Such loops can be nested, as shown in the example below.

**Syntax:** ***Next****(counterincrement)*

- *counterincrement* is the value to add to the counter pointed by the associated **For** statement. Negative increments can be used, allowing to decrement the counter.

The following examples, shared with the **Next** statement, are valid expressions:

```
Data cpt("1st loop counter", INT, {0})//

For(cpt, 0, >=, -5)
  // process to repeat as long as cpt>=-5
Next(-1)// decrements cpt

Data i("2nd loop counter", INT, {0})//
Data j("3rd loop counter", INT, {0})//

For(i, 0, <, 3)

  // process to repeat as long as i<3

  For(j, 0, >=, -5)
    // process to repeat as long as i<3 and j>=-5
  Next(-1) // decrements j

Next(1)// increments i

If(n, =, 1)
  Data imax("imax", INT, {1})//
Else
  If(n, =, 2)
    Data imax("imax", INT, {3})//
  Else
    Data imax("imax", INT, {4})//
  EndIf
EndIf

For(i, 0, <, imax)
  // process to repeat as long as I<imax
Next(1)//increments i
```

### 3.2.21: Plot:

**Object:** The Plot keyword allows to plot previously created Heelpoints, curves and GZ areas on the stability diagram.

**Syntax:** ***Plot*** *(variablename)*

- *variablename* is the internal name of the Curve, Area or Heelpoint to display on stability diagram. For obvious reasons, only these three kinds of variables can be plotted, provided that they have been created previously. The stability diagram is therefore empty when no **Plot** statement appears in the script.

The following examples are valid expressions:

```
Plot(gzmaxheel)// plots a vertical line at Heelpoint gzmaxheel
Plot(gzcurve)// plots GZ curve
Plot(gzareacurve)// plots GZ area curve
Plot(a_area)// shades a_area Area
Plot(b_area) // shades b_area Area
```

### 3.2.22: PlotGrid:

**Object:**. The Plotgrid keyword allows to select stability diagram's grid "densities".

**Syntax:** ***PlotGrid*** *(XMNEMONIC, YMNEMONIC)*

- *XMNEMONIC*, *YMNEMONIC* allows to select stability diagram's grid "densities" in x (horizontal) and y (vertical) directions.

**Mnemonics:**

- ***NO***: No grid is displayed in the diagram.
- ***LT***: A light grid is displayed in the diagram.
- ***HI***: A full grid is displayed in the diagram.

The following examples are valid expressions:

```
PlotGrid(NO, LT)
PlotGrid(LT, LT)
PlotGrid(HI, HI)
```

### 3.2.23: Print:

**Object:** The **Print** keyword allows printing a string as well as the name and value of any created data in the lower frame of stability diagram at a specified position.

**Syntax:** For a variable: ***Print****(variablename, line, column, optionalcolor)*
For a string: ***Print****(("string", line, column, optionalcolor)*
For a current string: ***Print****((MNEMONIC, line, column, optionalcolor)*

- *variablename* is the internal name of the data to print (Data, Heelpoint, Lever or Area).

- *string* is the string to print.

- *MNEMONIC* must be selected in this range:

  - *CURNTNAME*: Outputs current model's name.
  - *CURNTLOADING*: Outputs current loading's name or 'Current Loading' if undefined.
  - *CURNTDAMAGE*: Outputs current damage's name or 'Current Damage' if undefined .

- *line* is the printing line number (1 is the topmost, frame is automatically dimensionned in order to include all **Print**ed elements). *line* can be specified directly as a number or undirectly through an algebraic expression, which may be useful in printing loops.

- *column* is the printing column (counted in characters, 1 being the leftmost). *column* can be specified directly as a number or undirectly through an algebraic expression, which may be useful in printing loops (see last example).

- *optionalcolor* is the optional printing color among the following mnemonics:

  - *RED*: Prints in red
  - *BLUE*: Prints in blue
  - *GREEN*: Prints in green
  - *YELLOW*: Prints in yellow
  - *BLACK*: Prints in black (default)

The following examples are valid expressions:

```
Print(heq, 3, 2)
Print("Criterion: Intact River Boat", 1, 2)
Print CURNTLOADING, 1, 22)
Print CURNTDAMAGE, 2, 42)
If(ok)
  Print(x1, 12, {col})
Else
```

```
  Print(x1, 12, {col}, RED)
EndIf
```

### 3.2.24: PrintData:

**Object:** The **PrintData** keyword allows printing directly any data pointed by its hydrostatic mnemonic (see list in section 4.4) at a given heel angle at a specified position in the lower frame of stability diagram.

**Syntax:** ***PrintData****("variable user name", DATAMNEMONIC, heelangle, line, column)*

- *variable user name* is the user name to print for identifying the printed data.

- *DATAMNEMONIC* is the mnemonic of hydrostatic data to print to print (see list of available mnemonics in section 4.4).

- *heelangle* is the heel angle towhich the hydrostatic data pointed by *DATAMNEMONIC* must be printed.

- *line* is the printing line number (1 is the topmost, frame is automatically dimensionned in order to include all **Print**ed elements). *line* can be specified directly as a number or undirectly through an algebraic expression, which may be useful in printing loops.

- *column* is the printing column (counted in characters, 1 being the leftmost). *column* can be specified directly as a number or undirectly through an algebraic expression, which may be useful in printing loops (see last example).

**Mnemonics:**

Any hydrostatic mnemonic can be used (see list below, in section 4.4)

The following example is a valid expression:

```
PrintData("Vertical Center of Windage", VCW, heelangle, 1, 2)
```

### 3.2.25: RestrainDamageZone:

**Object:** The **RestrainDamageZone** keyword allows restraining longitudinally the damaged area in order to calculate the $p_i$ parameters needed by SOLAS 2009 probabilistic criterion. This restriction of damaged zone does not modify ship compartment's current status, but only allows controlling the range in which the $p_i$ subcalculation must be done.

**Syntax:** ***RestrainDamageZone****(aftsubdivisionindex, ndamagedcompartments)*

- *aftsubdivisionindex* defines the number of after watertight bulkhead limiting the damage. Constants, integer Data and complex albebraic expressions delimited by braces are accepted.

- *ndamagedcompartments* defines the number of contiguous damaged compartments (starting from after watertight bulkhead limiting the damage). Constants, integer Data and complex albebraic expressions delimited by braces are accepted.

The following examples are valid expressions (corresponding to the layout of most of the default criteria):

```
RestrainDamageZone(frstbulkheadnb, damageextent)
RestrainDamageZone(frstbulkheadnb, 1)
RestrainDamageZone({j+1}, {n-1})
```

### 3.2.26: ReportDamageData:

**Object:** When the damage system is calculated, reports and diagrams can be produced for analyzing the damage stability results. Among these outputs, a stability report is provided, listing all the data you may decide to printout in a one damage / line format. The **ReportDamageData** keyword allows the script to control the data displayed for each calculated damage.

**Syntax:** ***ReportDamageData****(value, column)*

- *data* is the data to print (Data, Heelpoint, Lever or Area).

- *column* is the printing column (counted in characters, 1 being the leftmost). *column* can be specified directly as a number or undirectly through an algebraic expression, which may be useful in printing loops (see last example).

The following examples are valid expressions:

```
ReportDamageData(pi, 11)
ReportDamageData(nui, {col})
```

### 3.2.27: SelectDamageFrom:

**Object:** The damage system generated for probabilistic calculations contain numerous damages grouped in different sets and subsets. Among each terminal subset, only one damage must be considered according to a given criterion (usually worst $s_i$). The **SelectDamageFrom** keyword allows defining how this calculated damage will be selected among its "brother" damages (the "damage branch" leaves).

**Syntax:** ***SelectDamageFrom****(criterion, comparison)*

- *criterion* is the variable to be tested for selecting the damage among its "brothers". The $s_i$ parameter is the used one for SOLAS 2009 calculations.

- *comparison* mnemonic defines how *criterion* has to be tested. The valid mnemonics are MIN and MAX.

The following example is the one used for SOLAS 2009 Calculation:

```
SelectDamageFrom(si, MIN)//Selects Damage Case with Minimum si among
siblings
```

### 3.2.28: SetBalanceHeel:

**Object:** The **SetBalanceHeel** statement allows specifying ship's equilibrium angle in order to initialize various depending data (GZ areas, MS, etc...). Depending on righting arm curve's complexity, ship's equilibrium angle is not always an obvious notion. This potentially complex situation has led to remove any automatic equilibrium calculation from MAAT Hydro's code since Rev. 7.0, transferring this calculation to the script in order to allow fully customizable definitions. As a counterpart of this freedom, MAAT Hydro must now necessarily be informed of the calculated equilibrium angle thanks to the **SetBalanceHeel** statement.

**Syntax:** ***SetBalanceHeel(****heelangle****)***

- *heelangle* defines the calculated equilibrium angle.

The following example is a valid expression:

```
Heelpoint heq("Ship Balance", INTERSECTION+, ox, gz, h0)
SetBalanceHeel(heq)
```

### 3.2.29: SetDamageIcon:

**Object:** When the script is executed for a given damage node, its initial white color is set from green to red, depending on its survivability and showing its calculated status. The **SetDamageIcon** keyword allows an STC script to set this node color according to the control parameter you supply and its associated range.

For example, $s_i$ can be used as the color control parameter, its range being therefore set to 0.0 (min value associated to red) and 1.0 (max value associated to green): Node's color will then change accordingly, representing damage node's survivability $s_i$ in a very intuitive way.

**Syntax:** ***SetDamageicon****(minredvalue, value, maxgreenvalue)*

- *minredvalue* is the minimum value of *value*, corresponding to a red damage node.

- *value* is the current value controlling damage node's color.

- *maxgreenvalue* is the maximum value of *value*, corresponding to a green damage node.

The following example is the one used for SOLAS 2009 Calculation:

```
SetDamageIcon(0.0, si, 1.0)// Scales Damage Icon Color
```

### 3.2.30: SetDamageProperty:

**Object:** When the script is executed for a given damage node, the calculation results are "encapsulated" in it, in order to remain instantly available for further use and analysis after the calculation sequence. The **SetDamageProperty** keyword allows encapsulating the calculated data listed below in each damage node at the end of each script execution.

**Syntax:** ***SetDamageProperty****(«name», n1, n2, type, pi, nui, si, dA, R, arc, gzx, fatalid)*

- *name* is the criterion name.

- *n1* specifies the N1 value.

- *n2* specifies the N2 value.

- *type* specifies the criterion type (PROBABILISTIC is the only mnemonic available for the moment).

- *pi* specifies the calculated $p_i$ value.

- *nui* specifies the calculated $nu_i$ value.

- *si* specifies the calculated $s_i$ value.

- *dA* specifies the calculated dA value.

- *R* specifies the calculated R value.

- *arc* specifies the calculated positive GZ arc value.

- *gzx* specifies the calculated max GZ value.

- *fatalid* identifies a special situation detected by script among the followings:

  - 0: No problem encountered in script.
  - 1: *frbd*: A freeboard line is immersed
  - 2: *opg*: An Opening 1 line is immersed
  - 3: *opg2*: An Opening 2 line is immersed
  - 4: *opg3*: An Opening 3 line is immersed
  - 5: *prfl*: A compartment's Overflow line is immersed by Progressive Flooding
  - 6: *mkln*: A Marking line is immersed
  - 7: *unwt*: A compartment's Unwettable line is immersed by Flooding
  - 8: *cpsz*: Damaged ship has no positive stability
  - 9: *fatl*: A Fatal compartment is damaged
  - 10: *sink*: Damaged ship has no positive buoyancy
  - 11: *fi.mm*: Mentions that $s_i$ is taken equal to the $s_{final}$ . $s_{mom}$ product.
  - 12: *inter*: Mentions that $s_i$ is taken equal to $s_{inter}$.

The following example is the one used for SOLAS 2009 calculation:

```
SetDamageProperty("SOLAS 2009", N1, N2, PROBABILISTIC, pi, nui, si, dA, R,
arc, gzx, 10)
```

### 3.2.31: SetIndexCoefficients:

**Object:** The **SetIndexCoefficients** keyword allows specifying the ratios used for testing the calculated probabilistic indexes when different of their default values (i.e. for SPS code), assuming that the probabilistic index tests are the followings:

A = alphas . As + alphap . Ap + alphal . Al > R
As > betas . R
Ap > betap . R
Al > betal . R

The default values are the followings:

alphas = 0.4
betas = 0.9
alphap = 0.4
betap = 0.9
alphal = 0.2
betal = 0.9

**Syntax:** ***SetIndexCoefficients****(alphas, betas, alphap, betap, alphal, betal)*

- *alphas, betas* are the As coefficients.

- *alphap, betap* are the Ap coefficients.

- *alphal, betal* are the Al coefficients.

The following example doesn't modify the default SOLAS 2009 coefficients:

```
SetIndexCoefficients(0.40, 0.90, 0.40, 0.90, 0.20, 0.90)
```

### 3.2.32: StartCSVTable:

**Object:** First of all, it is important to notice that any MAAT Hydro's 2D document may contain CSV exportable data, such 2D documents being simply identified by a red icon instead of a black one. As these CSV data "belong" to the document itself, they are then storable and reloadable with the project, therefore allowing any further export of the "encapsulated" CSV data. By default, CSV data are automatically added in certain 2D documents at generation time, but STC scripting allows a total redefinition of stability's CSV data by the user whenever necessary.

When present, 2D document's CSV data can be exported when necessary by selecting "CSV Export" in 2D viewport's menu (different header options are automaticalley proposed for the defaut non user-defined data). Nevertheless, it is important to stress that before being exported, these CSV data remain in MAAT Hydro's project and are therefore not yet available in an independent CSV file for an external use.

2D Document's CSV section being able to contain as many tables as necessary, the **StartCSVTable** keyword allows starting such tables, the **EndCSVTable** keyword (see below) being, conversely, necessary to close and validate it. There is no limitation on the amount of **StartCSVTable** / **EndCSVTable** delimitor pairs in a script.

**Syntax:** ***StartCSVTable****()*

**StartCSVTable** doesn't need any argument but must always match with an associated **EndCSVTable** for validating the table lines and cells defined between this delimitor pair.

The following example is a valid expression:

```
StartCSVTable()// creates a new table and its first CSV line
  AddCSVLineItem("This is an example ")// inserts text in the table
  NewCSVLine()// skips a line
EndCSVTable()// closes and validates the current table
```

### 3.2.33: Test:

**Object:** The **Test** keyword allows printing and testing any created data in the lower frame of stability diagram at a specified position, according to a test mnemonic and an associated threshold.

These tests are automatically displayed in black when compliant, otherwise in red.

Testing data is the goal of scripting, allowing to check if current stability diagram complies with a given regulation. Therfore, at least one **Test** keyword must appear in the script for allowing Max KG calculation, otherwise no critical KG will be determinable.

**Syntax:** ***Test****(variablename, TESTMNEMONIC, threshold, line, column)*

- *variablename* is the internal name of the data to test and print (Data, Heelpoint, Lever or Area).

- *TESTMNEMONIC* defines the test operation (see list of available mnemonics below).

- *threshold* is the value compared with *variablename* according to *TESTMNEMONIC*, in order to decide if tested data complies with the specified condition or not.

- *line* is the printing line number (1 is the topmost, frame is automatically fit in order to include all **Print**ed elements). *line* can be specified directly as a number or undirectly through an algebraic expression, which may be useful in printing loops.

- *column* is the printing column (counted in characters, 1 being the leftmost). *column* can be specified directly as a number or undirectly through an algebraic expression, which may be useful in printing loops (see last example).

**Mnemonics:**

- **>**: Test is succesfull when *variablename* is strictly greater than *threshold*.
- **>=**: Test is succesfull when *variablename* is equal or greater than *threshold*.
- **<**: Test is succesfull when *variablename* is strictly smaller than *threshold*.
- **<=**: Test is succesfull when *variablename* is equal or smaller than *threshold*.
- **=**: Test is succesfull when *variablename* is equal to *threshold*. This test is not reliable when applied to non integer data.
- **!=**: Test is succesfull when *variablename* is different of *threshold*. This test is not reliable when applied to non integer counters.

The following examples are valid expressions:

```
Test(phimax, >, phimom3, 7, 43)
Test(gzmax, >, 0.2, 7, 83)//

if(phif, <, phimax)
  Test(phif, >, phimom3, 8, 2)
```

```
  Test(gzf, >, 0.2, 8, 43)
                  Else
  Test(phif, >, phimom3, 8, 2)
                    Print(gzf, 8, 43)

EndIf

Test(a0max, >, limarea, 9, 43)
Test(gm0c, >, 0.15, 10, 83)

Test(phiwndpsg, <, 12.0, 11, 2)

If(lowinfbd, <, 1000.0)
  Test(lowinfbd, >, 0.1, 12, 60)//
      EndIf
```

### 3.2.34: Unit:

**Object:** The **Unit** keyword allows setting the current units in order to make the script constants independent of MAAT's current user units. It must therefore occur before setting any constant in the program, at the beginning of the script if possible.

**Syntax:** `Unit("unit")`

- *unit* is the string defining the units used for defining the script constants (see Units list below).

**Units:**

Available length units:

m: for meters (short format).
m+: for meters (long format).
dm: for decimeters (short format).
dm+: for decimeters (long format).
cm: for centimeters (short format).
cm+: for centimeters (long format).
mm: for millimeters (short format).
mm+: for millimeters (long format).
ft: for decimal feet (short format).
ft+: for decimal feet (long format).
in: for decimal inches (short format).
in+: for decimal inches (long format).

Available surface units:

m²: for square meters (short format).
m²+: for square meters (long format).
dm²: for square decimeters (short format).
dm²+: for square decimeters (long format).
cm²: for square centimeters (short format).
cm²+: for square centimeters (long format).
mm²: for square millimeters (short format).
mm²+: for square millimeters (long format).
sq.ft.: for square feet (short format).
sq.ft.+: for square feet (long format).
sq.in.: for square inches (short format).
sq.in.+: for square inches (long format).

Available volume units:

m3: for cubic meters (short format).
m3+: for cubic meters (long format).
dm3: for cubic decimeters (short format).

dm3+: for cubic decimeters (long format).
cm3: for cubic centimeters (short format).
cm3+: for cubic centimeters (long format).
mm3: for cubic millimeters (short format).
mm3+: for cubic millimeters (long format).
ft3: for cubic feet (short format).
ft3+: for cubic feet (long format).
in3: for cubic inches (short format).
in3+: for cubic inches (long format).

Available weight units:

t: for metric tons (short format).
t+: for metric tons (long format).
kg: for kilograms (short format).
kg+: for kilograms (long format).
g: for cubic grams (short format).
g+: for grams (long format).
lb: for pounds (short format).
lb+: for pounds (long format).

Available inertia units:

m4: for meters$^4$ (short format).
m4+: for meters$^4$ (long format).
dm4: for decimeters$^4$ (short format).
dm4+: for decimeters$^4$ (long format).
cm4: for centimeters$^4$ (short format).
cm4+: for centimeters$^4$ (long format).
mm4: for millimeters$^4$ (short format).
mm4+: for millimeters$^4$ (long format).
ft4: for feet$^4$ (short format).
in4: for inches$^4$ (short format).

Available lever area units:

m.rad: for meter.radians.

Available wind velocity units:

knots: for knots
m/s: for meter per second

The following examples are valid expressions:

```
Unit(m+)
Unit(m²+)
Unit(m3+)
Unit(m4+)
Unit(t+)
```

Unit(knots)
*Unit(m.rad)*

### 3.2.35: UserPrompt:

**Object:** The **UserPrompt** keyword allows displaying the info to keep in mind while using the criterion. Each **UserPrompt** command displays one line in the stability results dialog box, so that you must use n **UserPrompt** statements to display n lines. These reminders are usually set at the beginning of the script, preceding the user input specifications.

**Syntax:** ***Userprompt****("reminder")*

- *reminder* is the string defining the line to prompt in the reminders area.

The following examples are valid expressions:

```
UserPrompt("SOLAS 97 calculation requires:")
UserPrompt(" ")
UserPrompt("  - Damaged Compartment(s)")
UserPrompt("  - Freeboard descriptor(s) in 'Freeboard' layer")
UserPrompt("  - Opening descriptor(s) in 'Opening' layer")
UserPrompt(" ")
UserPrompt("Set Display Heel Range below in 'Min / Max Displayed Heel' fields")
```

### 3.2.36: WindGradient:

**Object:** The **WindGradient** keyword allows defining the wind strength gradient to be used for windage calculations. The **WindGradient** statement must therefore necessarily precede the the **CalculateWindageData** statement which integrates the windage data at a given heel angle (usually at ship's equilibrium) according to the previously defined wind data. The number of ($abscissa_i$, $ordinate_i$) pairs used for defining the gradient is not limited

**Syntax:** ***WindGradient***(*GRADIENTMNEMONIC, optional1, optional2, optionalabscissa1, optionalordinate1, ..., optionalabscissan, optionalordinaten*)

- *GRADIENTMNEMONIC* selects the gradient among predefined wind gradients (usually CONSTANT). The wind gradient is displayed on "Silhouette Analysis" outputs.

- *optional1* may be used:

  - For specifying a percentage of standard wind velocity when wind gradient must be adapted according to different velocities (see example below).

  - To create a user defined wind gradient by specifying its abscissa (OW, VCW are common, see 4.4 tables for full details). The following parameters are then necessary.

- *optional2* identifies the abscissa list's ordinate (PNM² is only available now, corresponding to N/m² pressures).

- *optionalabscissai* is the value of $i^{th}$ abscissa in the gradient table, according to *OPTIONALABSCISSAMNEMONIC*.

- *optionalordinatei* is the value of $i^{th}$ ordinate in the gradient table, according to *OPTIONALORDINATEMNEMONIC*.

**Mnemonics:**

Gradient mnemonics:

- CONSTANT: This mnemonic allows creating a constant wind gradient, which means that silhoutte will be integrated with a constant unitary pressure. When this constant wind pressure depends on the value pointed by *optional1*, a list of *optionalabscissai, optionalordinatei* pairs allow interpolating the constant pressure in a table (like for IMO A749) according to current abscissa value. The following examples are valid expressions of constant wind gradients:

```
WindGradient(CONSTANT, OW, PNM², 0, 540)
CalculateWindageData(heq)
```

```
WindGradient(CONSTANT, OW, PNM², 1, 316, 2, 386, 3, 429, 4, 460, 5, 485, 6, 504)
CalculateWindageData(heq)
```

- FR: This mnemonic allows creating a non constant wind gradient complying with French standard. By default, this gradient corresponds to a 100 knots wind at 10 m above floatation but this velocity can be weighed if necessary. As soon as a non constant wind gradient is defined, ship's silhouette is integrated with a non constant wind pressure according to height above floatation. The following examples are valid expressions:

```
WindGradient(FR)// Velocity = 100 knots by default…
CalculateWindageData(heq)
```

```
Unit(knots)
…
InputData wsmax("Wind velocity (Knots)" , SPD, 80)
…
Data ws("windspeed", SPD , {wsmax/100})
WindGradient(FR, ws)// Velocity can be weighed by ws if necessary
CalculateWindageData(h0)// Silhouette integrated at 0° heel
```

- US: This mnemonic allows creating a non constant wind gradient complying with US standard. By default, this gradient corresponds to a 100 knots wind at 10 m above floatation but this velocity can be weighed if necessary. As soon as a non constant wind gradient is defined, ship's silhouette is integrated with a non constant wind pressure according to height above floatation. The following examples are valid expressions:

```
WindGradient(US)// Velocity = 100 knots by default…
CalculateWindageData(heq)// Silhouette integrated at heq° heel
```

```
Unit(m/s)
…
InputData wsmax("Wind velocity (m/s)" , SPD, 80)// Default velocity: 80 kts @ 10m
…
Data ws("windspeed", SPD , {wsmax/100})
WindGradient(US, ws)// Velocity can be weighed by ws if necessary
CalculateWindageData(h0)
…
Data a("Mean Windage Area (m²)", AWD)
If(a, <, 0.001) Abort("No Silhouette Descriptor for Windage Calculation")
Data p("Mean Wind Pressure (N/m²)", MWP)// Mean wind pressure
Data ow("OW", OW) // CW height above floatation
Data z("Z", Z) // Directly gets Z = VCW-VCLA
Data zhd("ZHD", ZHD) // Directly gets Zhd = VCW-MDR/2
Data mmdr("MMDR (m)", MMDR) // Mean moulded draft
Data halfdraft("hdr", T/2) // Half mean moulded draft
Data vcw("VCW", VCW)
Data vcla("VCLA", VCL)
Data z1("Z' ", ABS, {vcw-vclp})
Data z2("Z''", ABS, {vcw-halfdraft})
```

Abscissa mnemonics:

Any hydrostatic mnemonic can be used, theoretically, but the most common are OW and VCW (see the complete list below, in section 4.4).

Ordinate mnemonics:

The only ordinate actually supported is PNM2, which corresponds to pressures in N/m².

## *3.2* Dimensional Mnemonics:

The available dimensional mnemonics are the followings:

| | |
|---|---|
| ABS | Real (no dimension) |
| ANG | Angle |
| CTR | Constraint |
| DNS | Density |
| GZA | GZ Area |
| INR | Inertia |
| INT | Integer (no dimension) |
| LEN | Length |
| MMT | Moment |
| PRS | Pressure |
| SRF | Surface |
| TIM | Time |
| VOL | Volume |
| WGT | Weight |
| SPD | Wind Velocity |

| | |
|---|---|
| Angle | ANG |
| Constraint | CTR |
| Density | DNS |
| GZ Area | GZA |
| Inertia | INR |
| Integer (no dimension) | INT |
| Length | LEN |
| Moment | MMT |
| Pressure | PRS |
| Real (no dimension) | ABS |
| Surface | SRF |
| Time | TIM |
| Volume | VOL |
| Weight | WGT |
| Wind Velocity | SPD |

## *3.3* Hydrostatic Data Mnemonics:

### 3.4.1: Hydrostatic Mnemonics and their corresponding Data:

| | |
|---|---|
| AAF° | Angle Aft Floatation (not yet calculated) |
| AFF° | Angle Fore Floatation (not yet calculated) |
| AK | Bilge or Bar Keel Area |
| ALP | Lateral Plane Area |
| AOV | Aft Overhang / AP |
| AS | Sail Area |
| AWD | Windage Area |
| AWDt | Total Windage Area |
| B² | B² Area |
| BDN | Min Bending Moment |
| BDX | Max Bending Moment |
| BIM | Breadth of Immersed Hull |
| BKEX | Outer Damage Breadth (controlled by RestrainDamageZone) |
| BKIN | Inner Damage Breadth (controlled by RestrainDamageZone) |
| BKMAX | Max Damage Breadth |
| BMd | Moulded Breadth |
| BKMIN | Min Damage Breadth |
| BML | Longit. Metacentric Radius |
| BMT | Transversz Metacentric Radius |
| BOA | Beam Over All |
| BS | Bs (Bmax) |
| BWL | Breadth of Waterline |
| CARGODENS | Viscous cargo's fictitious density used for FSM calculation |
| CARGOHEEL | Viscous cargo's free surface heel when different of waterplane's heel |
| CARGOW | Viscous cargo's weignt |
| CARGOX | Viscous cargo's C.G. x |
| CARGOY | Viscous cargo's C.G. y |
| CARGOZ | Viscous cargo's C.G. z |
| CB | Block Coefficient |
| CB2 | B² Area Ratio (area/width.depth) |
| CBBP | Block Coefficient (assuming that Lwl=LBP) |
| CMPXN | Min Damaged X (x1, independent of RestrainDamageZone) |
| CMPXX | Max Damaged X (x2, independent of RestrainDamageZone) |
| CP | Prismatic Coefficient |
| CPBP | Prismatic Coefficient (assuming that Lwl=LBP) |
| CW | Floatation Area Ratio (area /length.width) |
| CWS | Wetted Surface Ratio (WS/WSmin, with Wsmin=[18.PI.V²]^1/3) |
| DAMX1 | Minimum Damage X (x1, controlled by RestrainDamageZone) |
| DAP | AP Immersion |
| DFP | FP Immersion |
| DMP | MP Immersion |
| DECKZN0 | Minimum Damage Height (Hjnm-1) |
| DECKZN1 | Maximum Damage Height (Hjnm) |

| | |
|---|---|
| DGZtl | Liquid's Transverse GZ loss |
| DREF | Mean Moulded Draught at 0° |
| DryFbd | Dry Area Freeboard |
| DS | Ds |
| DSPL | Total Displacement |
| DSPm | Displacement (Moulded) |
| DWLZ | Dwl Height |
| FBMin | Min Free Board |
| FDFBDI | Foredeck freeboard corresponding to Ith pitch (if foredeck is present) |
| FLCG | Tank's Fluid CG position (x) |
| FMl | Tank's Fluid Longitudinal Metacentric Radius |
| FMt | Tank's Fluid Transverse Metacentric Radius |
| FNX | Floatation Normal (x) |
| FNY | Floatation Normal (y) |
| FNZ | Floatation Normal (z) |
| FOV | Fore Overhang / FP |
| FOX | Floatation Origin (x) |
| FOY | Floatation Origin (y) |
| FOZ | Floatation Origin (z) |
| FPC | Relative Tank's Filling (%) |
| FR0 | Reaction @ first extremity of Articulation 0 |
| FR1 | Reaction @ first extremity of Articulation 1 |
| FR2 | Reaction @ first extremity of Articulation 2 |
| FR3 | Reaction @ first extremity of Articulation 3 |
| FSML | Free Surf. Momt (Lngt.) |
| FSMT | Free Surface Moment (Trsv.) |
| FTCG | Tank's Fluid CG position (y) |
| FVCG | Tank's Fluid CG position (z) |
| FVOL | Tank's Fluid Volume |
| FWGT | Tank's Fluid Weight |
| GML | GMl, Longitudinal GM |
| GMLC | GMlc, Longitudinal GM (corrected) |
| GMT | GMt, Transverse GM |
| GMTC | GMtc, Transverse GM (corrected) |
| GMTFDK | GMt at foredeck's immersion (if foredeck is present) |
| GMTI | GMt value corresponding to Ith pitch (if foredeck is present) |
| GZ | Transverse Righting Arm(solid + liquid) |
| GZArea | GZ Area |
| GZL | Longitudinal Righting Arm (solid + liquid) |
| GZLFDK | GZl corresponding to foredeck's immersion (if present) |
| GZLI | GZl value corresponding to Ith pitch (if foredeck is present) |
| GZll | Longitudinal Righting Arm (liquid) |
| GZls | Longitudinal Righting Arm (solid) |
| GZLX | Max GZl value in the pitch range (if foredeck is present) |
| GZT | Transverse Righting Arm (solid + liquid) |
| GZtl | Transverse Righting Arm (liquid only) |
| GZts | Transverse Righting Arm (solid only) |
| H0 | Heel Angle in Articulation 0 referential |
| H1 | Heel Angle in Articulation 1 referential |

| | |
|---|---|
| H2 | Heel Angle in Articulation 2 referential |
| H3 | Heel Angle in Articulation 3 referential |
| HAK | Height Above K |
| HAM | Aft Mark Immersion |
| HBK | Height Below K |
| HBTM | Tank's Free Surface Height/Tank Bottom |
| Heel° | Heel in Degrees |
| HFM | Fore Mark Immersion |
| HGC | Tank's Fluid Corrected VCG" |
| HMDft | Half Mean Draft |
| HOA | Height Overall |
| HORG | Tank's Free Surface Height/Origin |
| HTOP | Tank's Free Surface Height/Top |
| INFLOW | Weight of water inflow in cargo tank |
| IMMPI | Mean draft corresponding to Ith pitch (if foredeck is present) |
| INFLOX | Cargo tank's water inflow C.G.x |
| INFLOY | Cargo tank's water inflow C.G.y |
| INFLOZ | Cargo tank's water inflow C.G.z |
| INTERFLOOD | Flooding Stage # |
| JAFTDMG | After damaged zone index (j) |
| JNUMDMG | Number of damaged zones (n) |
| KML | Height of Longitudinal Metacenter above Keel point (KMl) |
| KMT | Height of Transverse Metacenter above Keel point (KMt) |
| KN | KN |
| KZ | K Point Height |
| LAF | Aft Floatation pos. (x) |
| LAP | AP position |
| LB² | Center of B² (x) |
| LBDN | Min Bending Moment Position (x) |
| LBDX | Max Bending Moment Position (x) |
| LBM | Length Between Marks |
| LBP | Length Between Perpendiculars |
| LCB | Center of Buoyancy (x) |
| LCB/F | LCB / Fore Lwl ratio (%Lwl) |
| LCF | Center Floatation (x) |
| LCG | Total CG position (x) |
| LCGf | C.G. x position (liquid only) |
| LCGs | C.G. x position (solid only) |
| LCL | Center of Lateral Plane (x) |
| LCW | Center of Windage (x) |
| LCW-LCL | longitudinal distance between windage and lateral plane centers |
| LCW-LCL% | relative longit. dist. between windage & lateral plane centers (%LBP) |
| LFF | Fore Floatation pos. (x) |
| LFP | FP position |
| LGfl | C.G. x position (liquid only, corrected longit.) |
| LGft | C.G. x position (liquid only, corrected transv.) |
| LGtl | Total CG x position (corrected, longit.) |
| LGtt | Total CG x position (corrected, transv.) |
| LIM | Length of Immersed Hull |

| | |
|---|---|
| LI3 | Floatation’s Inertia ratio" |
| LMI | Longitdnal Moment of Inertia |
| LML | Longit. Metacenter (x) |
| LMP | MP position (x) |
| LMT | Transverse Metacenter (x) |
| LOA | Length Over All |
| LPR | Lateral Plane Ratio |
| LR0 | Reaction @ last extremity of Articulation 0 |
| LR1 | Reaction @ last extremity of Articulation 1 |
| LR2 | Reaction @ last extremity of Articulation 2 |
| LR3 | Reaction @ last extremity of Articulation 3 |
| LSHN | Min Shear Force Position (y) |
| LSHX | Max Shear Force Position (x) |
| LSMAX | Lsmax |
| LSMIN | Lsmin |
| LWL | Length of Waterline |
| MAXFBD | Max Freeboard |
| MAXOPG1 | Max Opening 1 Freeboard |
| MAXOPG2 | Max Opening 2 Freeboard |
| MAXOPG3 | Max Opening 3 Freeboard |
| MINDRY | Unwettable Lines Minimum Freeboard in damaged Compartments |
| MINFBD | Min Freeboard |
| MINOPG1 | Min Opening 1 Freeboard |
| MINOPG2 | Min Opening 2 Freeboard |
| MINOPG3 | Min Opening 3 Freeboard |
| MINMARK | Marking Lines Minimum Freeboard |
| MINPFF | Overflow Lines Min. Freeboard in intermediary flooded Compartments |
| MkFbd | Marked Area Freeboard |
| MMDR | Mean Moulded Draft (immersion of K point at MP) |
| MS | Residuary Righting Arm (MS) |
| MTC | Moment to Alter Trim 1cm |
| MWP | Mean Wind Pressure |
| NBODY | Number of bodies for multibody stability |
| NCMPTSABOVE | Number of Intact Compartments above Damage |
| NCOMP | Total Number of Compartrments |
| NDCMP | Number of Damaged Compartments (only) |
| NDITM | Number of Damaged Items (Compartments + Tanks) |
| NDmgUnfdblCpts | Damaged Unfloodable Cpts Number |
| NDTNK | Number of Damaged Tanks (only) |
| NICMP" | Number of Intact Compartments |
| NITNK | Number of Intact Tanks |
| NIITM | Number of Intact Items (Compartments + Tanks) |
| NOSTAB | Transverse Stability Indicator (0 if positive, 1 otherwise) |
| NSAFEDMG | Number of Damaged Fatal Compartments |
| NTNK | Number of Tanks |
| NWPLI | Number of Calculated Floatations |
| OG | Height of CG / Floatation (frozen liquids) |
| OGL | Height of CG / Floatation (corrected longit.) |
| OGT | Height of CG / Floatation (corrected transv.) |

| | |
|---|---|
| Op1FBMin | Min Opening1 Free Board |
| Op2FBMin | Min Opening2 Free Board |
| Op3FBMin | Min Opening3 Free Board |
| OS | Sail Center Height above Floatation |
| OVF | Overflow Height |
| OW | Heightof center of Windage above Floatation |
| PFF | Progressive Flooding Fbd |
| PHI | L/V^1/3 (slenderness coefficient) |
| PhiBP | Lwl/V^1/3 (assuming that Lwl=LBP) |
| PITCHI | Value of $I^{th}$ pitch in (°) |
| PITCHN | Number of calculated Pitches (when foredeck lines are present) |
| PTHK | Additional Plate Thickness |
| Ro | Water Density |
| SHN | Min Shear Force |
| SHX | Max Shear Force |
| SNDG | Tank's Sounding |
| T0 | Trim in Articulation 0 referential |
| T1 | Trim in Articulation 1 referential |
| T2 | Trim in Articulation 2 referential |
| T3 | Trim in Articulation 3 referential |
| T/2 | Half Draft |
| TA | AP Immersion |
| TAM | Aft Mark Immersion |
| TB² | Center of B² (y) |
| TCB | Center of Buoyancy (y) |
| TCF | Center Floatation (y) |
| TCG | CG position (y) |
| TCGf | C.G. y position (liquid only) |
| TCGs | C.G. y position (solid only) |
| TF | FP Immersion |
| TFM | Fore Mark Immersion |
| TGfl | C.G. y position (liquid only, corrected longit.) |
| TGft | C.G. y position (liquid only, corrected transv.) |
| TGtl | Total CG y position (corrected, longit.) |
| TGtt | Total CG y position (corrected, transv.) |
| TH0 | Theta0 Angle of Articulation 0 |
| TH1 | Theta1 Angle of Articulation 1 |
| TH2 | Theta2 Angle of Articulation 2 |
| TH3 | Theta3 Angle of Articulation 3 |
| TM | MP Immersion |
| TM0 | MP Height in Articulation 0 referential |
| TM1 | MP Height in Articulation 1 referential |
| TM2 | MP Height in Articulation 2 referential |
| TM3 | MP Height in Articulation 3 referential |
| TMI | Transversal Moment of Inertia |
| TML | Longit. Metacenter (y) |
| TMT | Transverse Metacenter (y) |
| TOA | Draft at waterline |
| TPC | Tons per cm |

| | |
|---|---|
| Trim | Trim (height) |
| Trim° | Trim (angle in degrees) |
| TRIMFDK | Foredeck's immersion trim (if foredeck) |
| TRIMX | Foredeck's max scanning trim (if foredeck) |
| ULG | Tank's Ullage |
| Uml | Unitary Moment (Longit.) |
| Umt | Unitary Moment (Transv.) |
| Umtc | Transverse Unitary Moment |
| UWP | Unitary Wind Pressure |
| VB² | Center of B² (z) |
| VCB | Center of Buoyancy (z) |
| VCF | Center Floatation (z) |
| VCG | Total CG position (z) |
| VCGf | C.G. z position (liquid only) |
| VCGs | C.G. z position (solid only) |
| VCL | Center of Lateral Plane (z) |
| VCW | Center of Windage (z) |
| VGfl | C.G. z position (liquid only, corrected longit.) |
| VGft | C.G. z position (liquid only, corrected transv.) |
| VGtl | Total CG z position (corrected, longit.) |
| VGtt | Total CG z position (corrected, transv.) |
| VML | Longit. Metacenter (z) |
| VMT | Transverse Metacenter (z) |
| VOL | Volume |
| VOLm | Volume (Moulded) |
| Wamp | Wave Amplitude |
| WCr | Wave Crest position (x) |
| Wfix | Weight (solid only) |
| WGT | Total Weight |
| WLgt | Wave Length |
| Wliq | Weight (liquid only)", |
| WPA | Waterplane Area |
| WSA | Wetted Surface |
| WVA | Wave Amplitude |
| WVD | Wave Direction (not used yet) |
| WVL | Wave Length |
| WVP | Wave Phase |
| XAM | Aft Mark position (x) |
| XFM | Fore Mark position (x) |
| Z | vertical windage/ lateral plane centers distance (freeboard only) |
| ZAM | Aft Mark position (z) |
| ZFM | Fore Mark position (z) |
| ZHD | vertical distance between center of windage and half draft (VCW-T/2) |
| ZML | Longitudinal Metacentric Height |
| ZMLC | Longitudinal Metacentric Height (corrected) |
| ZMT | Transverse Metacentric Height |
| ZMTC | Transverse Metacentric Height (corrected) |
| Zt | Vertical distance between total windage and lateral plane centers (sail + freeboard) |

### 3.4.2: Hydrostatic data and their corresponding mnemonics:

| | |
|---|---|
| Aft Mark position (z) | ZAM |
| Additional Plate Thickness | PTHK |
| Aft Floatation pos. (x) | LAF |
| Aft Mark Immersion | HAM |
| Aft Mark Immersion | TAM |
| Aft Mark position (x) | XAM |
| Aft Overhang / AP | AOV |
| After damaged zone index (j) | JAFTDMG |
| Angle Aft Floatation (not yet calculated) | AAF° |
| Angle Fore Floatation (not yet calculated) | AFF° |
| AP Immersion | TA |
| AP position | LAP |
| B² Area | B² |
| B² Area Ratio (area/width.depth) | CB2 |
| Beam Over All | BOA |
| Bilge or Bar Keel Area | AK |
| Block Coefficient | CB |
| Block Coefficient (assuming that Lwl=LBP) | CBBP |
| Breadth of Immersed Hull | BIM |
| Breadth of Waterline | BWL |
| Bs (Bmax) | BS |
| Cargo tank' water inflow weight | INFLOW |
| Cargo tank' water inflow C.G.x | INFLOX |
| Cargo tank' water inflow C.G.y | INFLOY |
| Cargo tank' water inflow C.G.z | INFLOZ |
| C.G. y position (liquid only) | TCGf |
| C.G. y position (liquid only, corrected longit.) | TGfl |
| C.G. y position (liquid only, corrected transv.) | TGft |
| C.G. y position (solid only) | TCGs |
| C.G. z position (liquid only) | VCGf |
| C.G. z position (liquid only, corrected longit.) | VGfl |
| C.G. z position (liquid only, corrected transv.) | VGft |
| C.G. z position (solid only) | VCGs |
| Center Floatation (x) | LCF |
| Center Floatation (y) | TCF |
| Center Floatation (z) | VCF |
| Center of B² (x) | LB² |
| Center of B² (y) | TB² |
| Center of B² (z) | VB² |
| Center of Buoyancy (x) | LCB |
| Center of Buoyancy (y) | TCB |
| Center of Buoyancy (z) | VCB |
| Center of Lateral Plane (x) | LCL |
| Center of Lateral Plane (z) | VCL |
| Center of Windage (x) | LCW |

| | |
|---|---|
| Center of Windage (z) | VCW |
| CG position (y) | TCG |
| Damaged Unfloodable Cpts Number | NDmgUnfdblCpts |
| Displacement (Moulded) | DSPm |
| Draft at waterline | TOA |
| Dry Area Freeboard | DryFbd |
| Ds | DS |
| Dwl Height | DWLZ |
| Floatation Area | AWL |
| Floatation Area Ratio (area /length.width) | CW |
| Floatation Normal (x) | FNX |
| Floatation Normal (y) | FNY |
| Floatation Normal (z) | FNZ |
| Floatation Origin (x) | FOX |
| Floatation Origin (y) | FOY |
| Floatation Origin (z) | FOZ |
| Floatation's Inertia ratio" | LI3 |
| Flooding Stage # | INTERFLOOD |
| Fluid Density | SG |
| Foredeck's freeboard corresponding to Ith pitch (if foredeck is present) | FDFBDI |
| Foredeck's immersion trim (if foredeck) | TRIMFDK |
| Foredeck's max scanning trim (if foredeck) | TRIMX |
| Fore Floatation pos. (x) | LFF |
| Fore Mark Immersion | HFM |
| Fore Mark Immersion | TFM |
| Fore Mark position (x) | XFM |
| Fore Mark position (z) | ZFM |
| Fore Overhang / FP | FOV |
| FP Immersion | TF |
| FP position | LFP |
| Free Surf. Momt (Lngt.) | FSML |
| Free Surface Moment (Trsv.) | FSMT |
| GMl, Longitudinal GM | GML |
| GMlc, Longitudinal GM (corrected) | GMLC |
| GMt, Transverse GM | GMT |
| GMtc, Transverse GM (corrected) | GMTC |
| GMt at foredeck's immersion (if foredeck is present) | GMTFDK |
| GMt value corresponding to Ith pitch (if foredeck is present) | GMTI |
| GZ Area | GZArea |
| GZl corresponding to foredeck's immersion (if present) | GZLFDK |
| GZl corresponding to Ith pitch (if foredeck is present) | GZLI |
| Half Draft | T/2 |
| Half Mean Draft | HMDft |
| Heel in Degrees | Heel° |
| Heel Angle in Articulation 0 referential | H0 |
| Heel Angle in Articulation 1 referential | H1 |
| Heel Angle in Articulation 2 referential | H2 |
| Heel Angle in Articulation 3 referential | H3 |
| Height Above K | HAK |

| | |
|---|---|
| Height Below K | HBK |
| Height of CG / Floatation (corrected longit.) | OGL |
| Height of CG / Floatation (corrected transv.) | OGT |
| Height of CG / Floatation (frozen liquids) | OG |
| Height of Longitudinal Metacenter above Keel point (KMl) | KML |
| Height of Transverse Metacenter above Keel point (KMt) | KMT |
| Height Overall | HOA |
| Heightof center of Windage above Floatation | OW |
| Inner Damage Breadth (controlled by RestrainDamageZone) | BKIN |
| K Point Height | KZ |
| KN | KN |
| L/V^1/3 (slenderness coefficient) | PHI |
| Lateral Plane Area | ALP |
| Lateral Plane Ratio | LPR |
| LCB / Fore Lwl ratio (%Lwl) | LCB/F |
| Length Between Marks | LBM |
| Length Between Perpendiculars | LBP |
| Length of Immersed Hull | LIM |
| Length of Waterline | LWL |
| Length Over All | LOA |
| Liquid's Transverse GZ loss | DGZtl |
| Longit. Metacenter (x) | LML |
| Longit. Metacenter (y) | TML |
| Longit. Metacenter (z) | VML |
| Longit. Metacentric Radius | BML |
| Longitdnal Moment of Inertia | LMI |
| longitudinal distance between windage and lateral plane centers | LCW-LCL |
| Longitudinal Metacentric Height | ZML |
| Longitudinal Metacentric Height (corrected) | ZMLC |
| Longitudinal Righting Arm (liquid) | GZll |
| Longitudinal Righting Arm (solid + liquid) | GZL |
| Longitudinal Righting Arm (solid) | GZls |
| Lsmax | LSMAX |
| Lsmin | LSMIN |
| Lwl/V^1/3  (assuming that  Lwl=LBP) | PhiBP |
| Marked Area Freeboard | MkFbd |
| Marking Lines Minimum Freeboard | MINMARK |
| Max Bending Moment | BDX |
| Max Bending Moment Position (x) | LBDX |
| Max Damage Breadth | BKMAX |
| Max Damaged X (x2, independent of RestrainDamageZone) | CMPXX |
| Max Free Board | MAXFBD |
| Max Opening1 Free Board | MAXOPG1 |
| Max Opening2 Free Board | MAXOPG2 |
| Max Opening3 Free Board | MAXOPG3 |
| Max GZl value in the pitch range (if foredeck is present) | GZLX |
| Max Shear Force | SHX |
| Max Shear Force Position (x) | LSHX |
| Maximum Damage Height (Hjnm) | DECKZN1 |

| | |
|---|---|
| Maximum Damage X (X2, controlled by RestrainDamageZone) | DAMX2 |
| Mean Draft (area of lateral plane / lwl) | MDR |
| Mean Moulded Draft (immersion of K point at MP) | MMDR |
| Mean Moulded Draught at 0° | DREF |
| Mean Moulded draft corresponding to Ith pitch (if foredeck is present) | IMMPI |
| Mean Wind Pressure | MWP |
| Min Bending Moment | BDN |
| Min Bending Moment Position (x) | LBDN |
| Min Damage Breadth | BKMIN |
| Min Damaged X (x1, independent of RestrainDamageZone) | CMPXN |
| Min Free Board | MINFBD |
| Min Opening Free Board | MINOPG |
| Min Opening1 Free Board | MINOPG1 |
| Min Opening2 Free Board | MINOPG2 |
| Min Opening3 Free Board | MINOPG3 |
| Min Shear Force | SHN |
| Min Shear Force Position (y) | LSHN |
| Minimum Damage Height (Hjnm-1) | DECKZN0 |
| Minimum Damage X (X1, controlled by RestrainDamageZone) | DAMX1 |
| Moment to Alter Trim 1cm | MTC |
| Moulded Breadth | BMd |
| MP Height in Articulation 0 referential | TM0 |
| MP Height in Articulation 1 referential | TM1 |
| MP Height in Articulation 2 referential | TM2 |
| MP Height in Articulation 3 referential | TM3 |
| MP Immersion | TM |
| MP position (x) | LMP |
| Number of bodies for multibody stability | NBODY |
| Number of Calculated Floatations | NWPLI |
| Number of Damaged Compartments | NDCMP |
| Number of Damaged Tanks | NDTNK |
| Number of Damaged Items (Compartments + Tanks) | NDITM |
| Number of Damaged Fatal Compartments | NSAFEDMG |
| Number of Damaged zones (n) | JNUMDMG |
| Number of Intact Compartments | NICMP |
| Number of Intact Compartments above Damage | NCMPTSABOVE |
| Number of Intact Tanks | NITNK |
| Number of Intact Items (Compartments + Tanks) | NIITM |
| Number of calculated Pitches (when foredeck lines are present) | PITCHN |
| Number of Tanks | NTNK |
| Outer Damage Breadth (controlled by RestrainDamageZone) | BKEX |
| Overflow Height | OVF |
| Overflow Lines Min. Freeboard in intermediary flooded Compartments | MINPFF |
| Prismatic Coefficient | CP |
| Prismatic Coefficient (assuming that Lwl=LBP) | CPBP |
| Progressive Flooding Fbd | PFF |
| Reaction @ first extremity of Articulation 0 | FR0 |
| Reaction @ first extremity of Articulation 1 | FR1 |

| | |
|---|---|
| Reaction @ first extremity of Articulation 2 | FR2 |
| Reaction @ first extremity of Articulation 3 | FR3 |
| Reaction @ last extremity of Articulation 0 | LR0 |
| Reaction @ last extremity of Articulation 1 | LR1 |
| Reaction @ last extremity of Articulation 2 | LR2 |
| Reaction @ last extremity of Articulation 3 | LR3 |
| Relative longit. dist. between windage and lateral plane centers (%LBP) | LCW-LCL% |
| Relative Tank's Filling (%) | FPC |
| Residuary Righting Arm (MS) | MS |
| Sail Area | AS |
| Sail Center Height above Floatation | OS |
| Tank's Fluid CG position (x) | FLCG |
| Tank's Fluid CG position (y) | FTCG |
| Tank's Fluid CG position (z) | FVCG |
| Tank's Fluid Corrected VCG" | HGC |
| Tank's Fluid Longitudinal Metacentric Radius | FMl |
| Tank's Fluid Transverse Metacentric Radius | FMt |
| Tank's Fluid Volume | FVOL |
| Tank's Fluid Weight | FWGT |
| Tank's Free Surface Height/Origin | HORG |
| Tank's Free Surface Height/Tank Bottom | HBTM |
| Tank's Free Surface Height/Top | HTOP |
| Tank's Sounding | SNDG |
| Tank's Ullage | ULG |
| Theta0 Angle of Articulation 0 | TH0 |
| Theta1 Angle of Articulation 1 | TH1 |
| Theta2 Angle of Articulation 2 | TH2 |
| Theta3 Angle of Articulation 3 | TH3 |
| Tons per cm of immersion | TPC |
| Total CG position (x) | LCG |
| Total CG position (y) | TCG |
| Total CG position (z) | VCG |
| Total CG x position (corrected, longit.) | LGtl |
| Total CG x position (corrected, transv.) | LGtt |
| Total CG y position (corrected, longit.) | TGtl |
| Total CG y position (corrected, transv.) | TGtt |
| Total CG z position (corrected, longit.) | VGtl |
| Total CG z position (corrected, transv.) | VGtt |
| Total Displacement | DSPL |
| Total Number of Compartments | NCOMP |
| Total Weight | WGT |
| Total Windage Area | AWDt |
| Transversal Moment of Inertia | TMI |
| Transverse Metacenter (x) | LMT |
| Transverse Metacenter (y) | TMT |
| Transverse Metacenter (z) | VMT |
| Transverse Metacentric Height | ZMT |
| Transverse Metacentric Height (corrected) | ZMTC |
| Transverse Righting Arm (liquid only) | GZtl |

| | |
|---|---|
| Transverse Righting Arm (solid + liquid) | GZT |
| Transverse Righting Arm (solid only) | GZts |
| Transverse Righting Arm(solid + liquid) | GZ |
| Transverse Stability Indicator (0 if positive, 1 otherwise) | NOSTAB |
| Transverse Unitary Moment | Umtc |
| Transverse Metacentric Radius | BMT |
| Trim (angle in degrees) | Trim° |
| Trim (height) | Trim |
| Trim in Articulation 0 referential | T0 |
| Trim in Articulation 1 referential | T1 |
| Trim in Articulation 2 referential | T2 |
| Trim in Articulation 3 referential | T3 |
| Unitary Moment (Longit.) | Uml |
| Unitary Moment (Transv.) | Umt |
| Unitary Wind Pressure | UWP |
| Unwettable Lines Minimum Freeboard in damaged Compartments | MINDRY |
| Value of $I^{th}$ pitch in (°) | PITCHI |
| vertical distance between center of windage and half draft (VCW-T/2) | ZHD |
| Vertical distance between total windage and lateral plane centers (sail + freeboard) | Zt |
| Vertical distance between windage and lateral plane centers (freeboard only) | Z |
| Viscous cargo's fictitious density used for FSM calculation | CARGODENS |
| Viscous cargo's free surface heel when different of waterplane's heel | CARGOHEEL |
| Viscous cargo's weight | CARGOW |
| Viscous cargo's C.G. x | CARGOX |
| Viscous cargo's C.G. y | CARGOY |
| Viscous cargo's C.G. z | CARGOZ |
| Water Density | Ro |
| Wave Direction (not used yet) | WVD |
| Wave Length | WLgt |
| Wave Length | WVL |
| Wave Phase | WVP |
| Weight (liquid only)", | Wliq |
| Weight (solid only) | Wfix |
| Wetted Surface | WS |
| Wetted Surface Ratio (WS/WSmin, with Wsmin=[18.PI.V²]^1/3) | CWS |
| Windage Area | AWD |