Tutorial 2: Objects and TeLiTabs

Tutorial 2 Objects and TeLiTabs	Learning goals	Prior knowledge	
	1. Learn about the TeLiTab data format 2. Using an object as a function 3. Creating a multiple case solution	1. Interface 2. Tutorial 1: Getting Started	

1 Objective

In this tutorial, you will create the shape of a ship's waterline in a table of frame numbers and relative, dimensionless widths. The dataset containing variations of the main dimensions of a ship from tutorial 1 is stored in a data object *Ships* in the dataset of the knowledge base, so it can be easily used in further calculations. This dataset combined with the dimensionless width table is used to calculate the width at the waterline at every other frame number for every variation.

> ⚠ Note that the word frame is used here as a ship's frame, not as a frame in Quaestor.

Start

For this tutorial, the knowledgebase from tutorial 1 is used. You can either use your own (verified) knowledgebase, or download it here: [Tutorial 2 Start]

2 Background information: TeLiTab data model and Quaestor object

TeLiTab is a term that will be used quite often in this tutorial. Therefore, an elaboration on the subject is in order in this very early stage for the tutorials.

TeLiTab is an abbreviation of **Text**, **List**, **Table** (see TeLiTab for more detail). It is the standard format in which data is stored inside Quaestor, the easiest way to use data inside a knowledge based systems and the easiest way to exchange data with external applications (see also Use of external or satellite programs). An example of a telitab is given below, the black text to the right is comment and is not present in the actual telitab.

```
1                       Number of list items in primary Telitab
"Ship"                  Name of List item
   {                    Opening secondary Telitab
   3                    Number of list items
   "Lpp" 100            List item 1, name + value
   "B" 20               List item 2 name + value
   "Resistance"         List item 3, which is a Telitab
      {                 Opening tertiary Telitab
      0                 Number of list items (in this case zero)
      2 "R" "V"         Number of table parameters and their names
      "1" 1000 10       Dataset per case
      "2" 1200 11
      "3" 1300 12
      "4" 1400 13
      "5" 1500 14
      "6" 1600 15
      }                 Closing tertiary Telitab
   }                    Closing secondary Telitab
                        Primary Telitab does not need to be closed
```

Every level of a TeLiTab contains list items and an optional table. The table can be omitted, but if there are no list items present at a certain level, a '0' is required (like above in the resistance telitab). Parameters and case numbers are written between double quotes (""). Numerical values are not written between quotes, strings that contain spaces are written between double quotes. For all syntax aspects please go to: TeLiTab

A **Quaestor Object** contains a set of data, either static (only data, e.g. a speed power curve or a list of components) or dynamic, and is represented as TeLiTab. An object can operate as a computational model, (requesting input and providing output) fulfilling the role of a function or subroutine in an assembled model (a 'Solution' in the Quaestor workbase).

The use of the TeLiTab format and Quaestor objects is covered in this tutorial.

3 Creating a table with the coordinates of the waterline

In order to create the geometry of your ship, you will need an object that contains frame numbers and relative, dimensionless widths. To keep the knowledgebase arranged logically, create a new class:

- In the Knowledge Browser, create a new class named `Geometry` directly below *Top Goals/Undefined*.

Creating classes is covered in tutorial 1.

The ship is defined mathematically, which makes it convenient to generate. The relation between the frame number (0 to 20) and relative width is:

```
Rel_B = 0.5 - 0.005 * (Frame - 10)^2
```

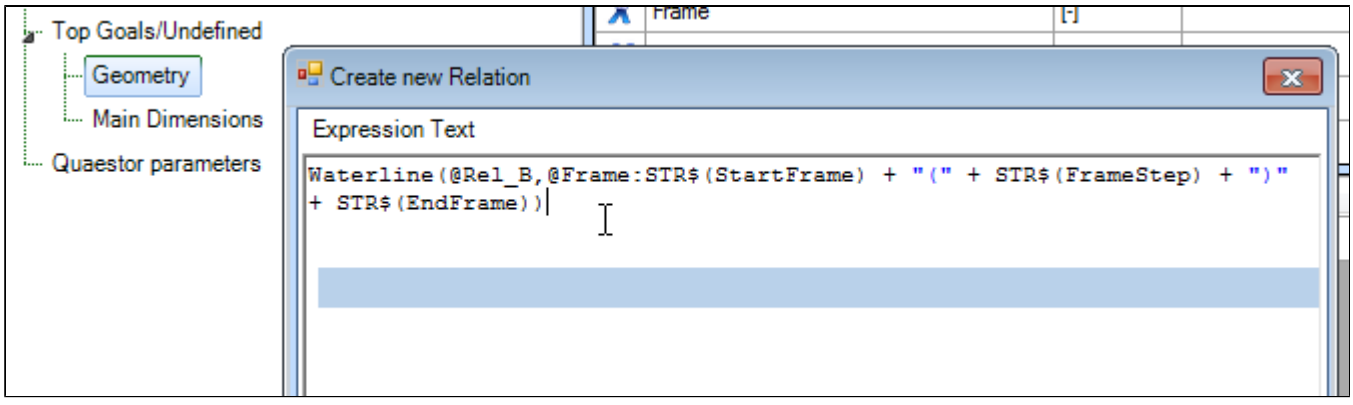
The shape of the waterline is a polynomial (a parabola). You can enter this function in the Geometry class in the same way as in tutorial 1.

- Add the relation above to the class `Geometry`. The parameters are automatically created when entering a relation. `Frame` should be listed as a *user defined (VR)* parameter and `Rel_B` should be *System defined (SYS)*. Don't forget the dimensions (`Rel_B` and `Frame` are both dimensionless), and a proper reference for the parameters.

Using the function defined above, put the shape in an object.

- Add the following relation to the class `Geometry`:

```
Waterline(@Rel_B,@Frame:STR$(StartFrame) + "(" + STR$(FrameStep) + ")" + STR$(EndFrame))
```



Upon saving the relation, the following questions will be asked:

1. You will be asked whether `StartFrame` is an Object. Say no;
2. Hereafter you will be asked whether `StartFrame` is a Goal parameter. Say no, because you want to provide this as input for the `Waterline` object;
3. These questions are also asked for `FrameStep` and `EndFrame`. Say no to the Object question and no to the Goal parameter question. These values are input to the object.

- The three new parameters all are dimensionless and should be of the *VR* type.

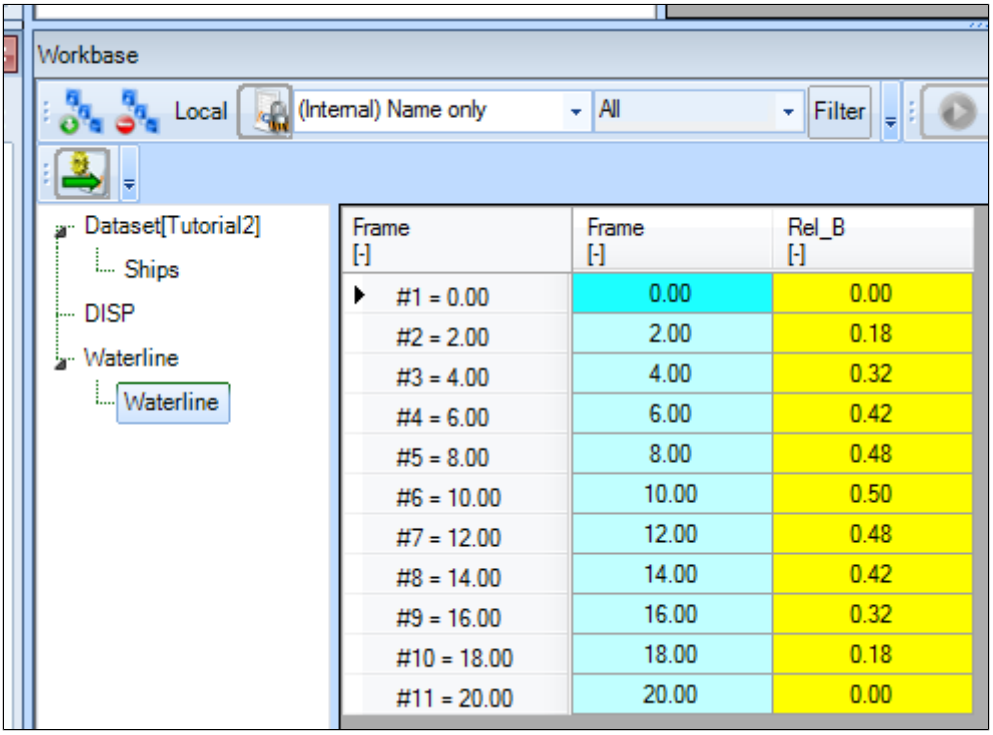
The syntax of this relation defines a Function that creates an object `Waterline` including `Rel_B` and `Frame` the moment it is initiated (when you do a calculation). The first parameter, `@Rel_B`, makes sure `Rel_B` is the top goal (calculated value). The second parameter, `@Frame`, makes sure `Frame` is the input for the function, the expression following `@Frame: STR$(StartFrame) + "(" + STR$(FrameStep) + ")" + STR$(EndFrame)` defines what should be asked as input to create the range input within the `Waterline` object.

The general syntax used for `Frame` is `@ParameterName:"Input"`. By using the intrinsic STR$ function with the parameters, you can give range input to the object, while for Quaestor it appears as normal single value input. This solves the limitation that you cannot give range input while building a solution. The `@` in front of the parameters is to indicate that `Rel_B` and `Frame` are to be calculated in the object. See also QuaestorSyntax.

Note that just after you have created the expression, `Waterline` appears as a new parameter in the knowledge base with dimension object *(Obj)*. So, in Quaestor, a parameter can be seen as an object and objects can be used as functions to perform calculations in.

- Select `Waterline` as top goal by double clicking it, and start a solution. Click *Yes* when asked if a new solution should be created. Quaestor will ask for the `StartFrame`, `EndFrame` and `FrameStep`. Please note that the parameters are requested in alphabetical order. Enter for the `StartFrame` 0, for the `FrameStep` 2 and `EndFrame` 20, and click *Next*.

The result will look like this:



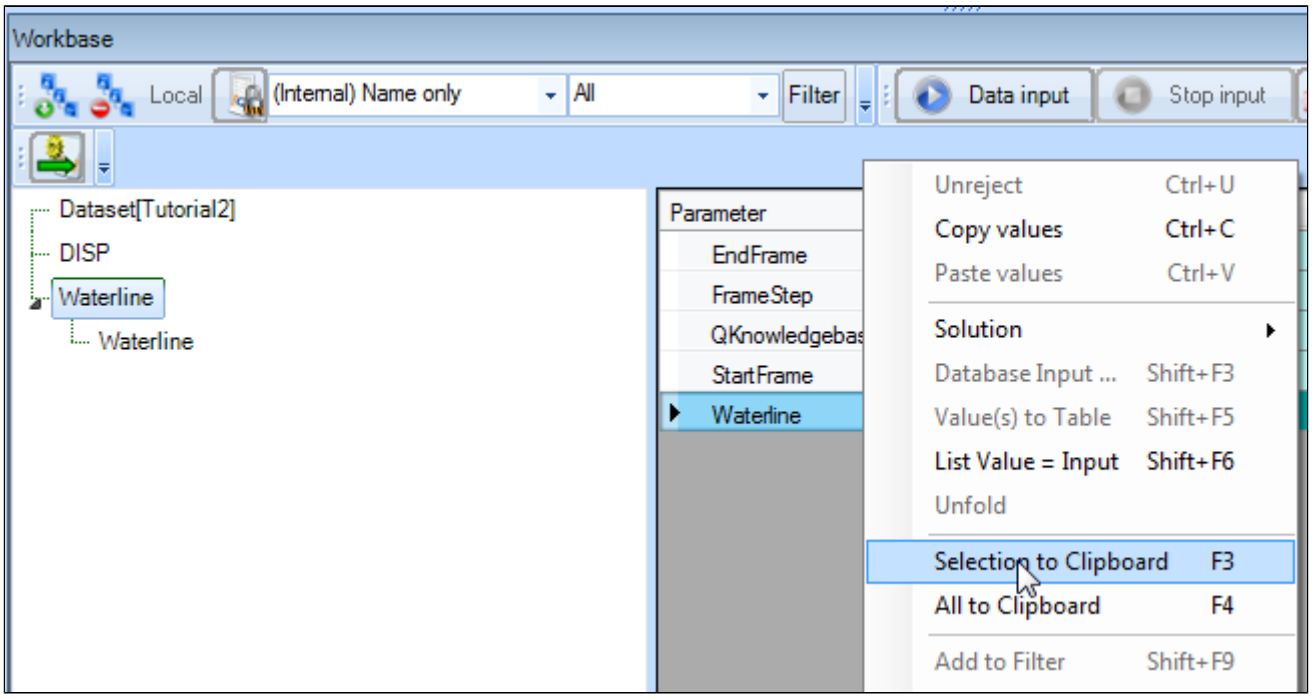
The solution is the object `Waterline`, which is filled with values for `Rel_B` for every case of `Frame`.

By using the `@ORDER` attribute in the `Data` field property, you can manipulate the input order (provide `@ORDER:1` to `StartFrame`, `@ORDER:2` to `Frame Step` and `@ORDER:3` to `EndFrame`). Furthermore, please note that you cannot select Waterline as top goal when you are into the Waterline object in one of the solutions. The reason is that Quaestor assumes you want to add a parameter to this level and does not allow you to do so.

Please also note that inside the Waterline node, the dimension of Waterline is stated a Function, which is the correct conclusion drawn by Quaestor.

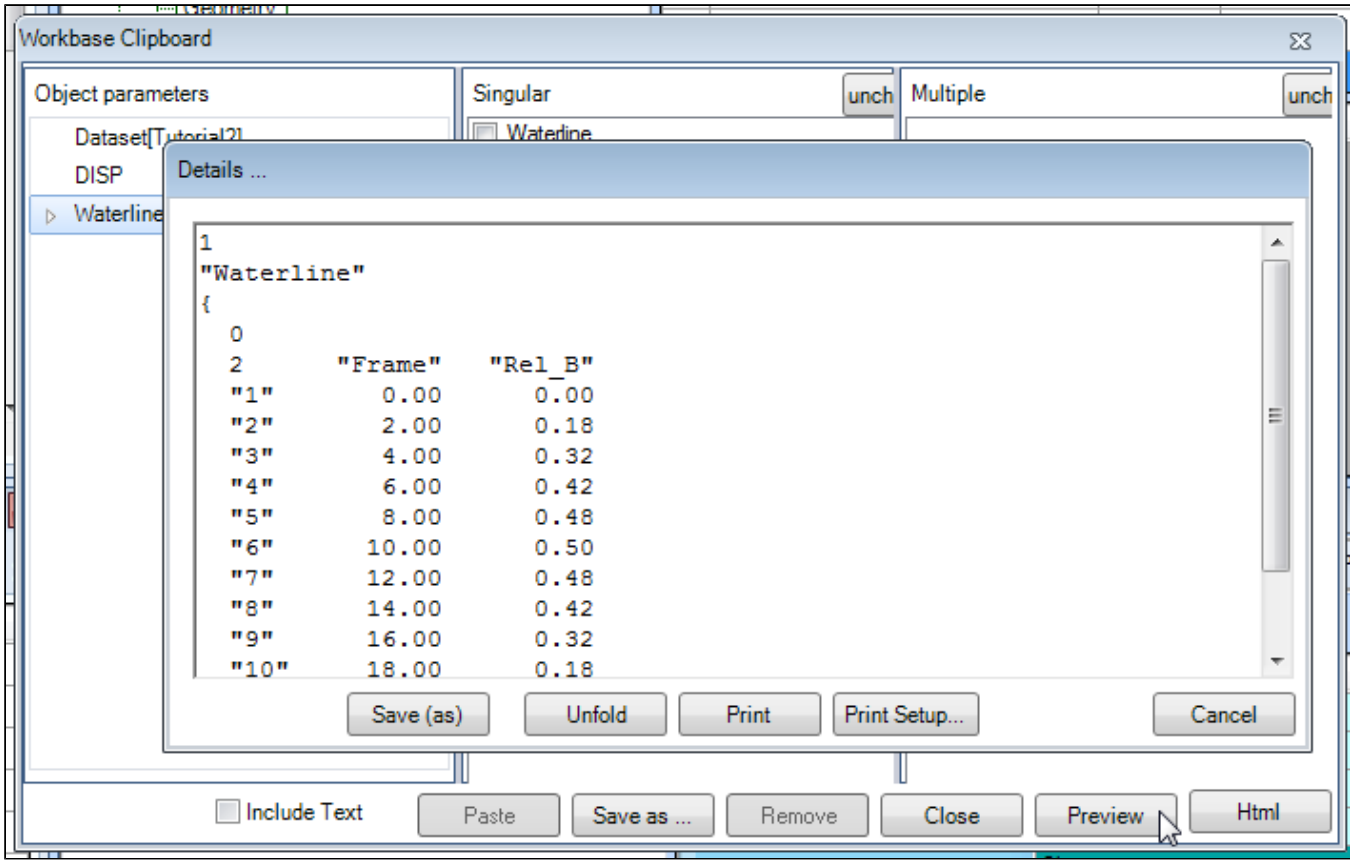
4 Telitab of the Waterline object

In the solution *Waterline*, select the cell containing *Object,* right-click and select *Selection to Clipboard*, or press *F3*.



The Workbase Clipboard is opened.

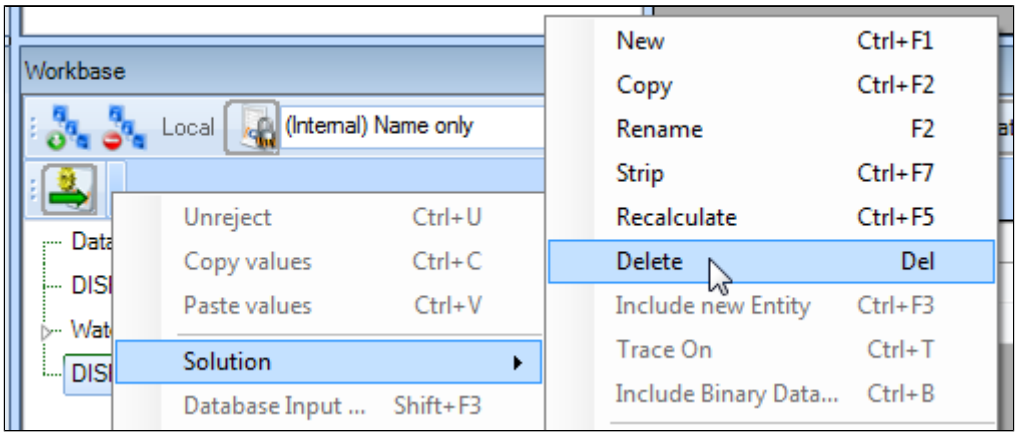
- Click on *Preview* and the contents of the clipboard are shown. These contents consist of the contents of the `Waterline` object: Click *Cancel* after this.



The TeLiTab format used to describe the contents of the object is clearly visible. There is 1 list item, named `Waterline.` This is a telitab (2nd level), containing 0 **list items** and a **table** with 2 parameters, *Frame* and *Rel_B*. The **case numbers** are shown between double quotes ("").

5 Managing solutions

To clean up the Workbase from some of your trials, select the solution in the Workbase you want to delete, right-click and select *Solution -> Delete* or press *Del*. Click *yes* to confirm. The solution has been deleted from the workbase.



Manipulating solutions only works when Quaestor is not in calculation mode. Press *Stop Input* until this button becomes disabled to exit calculation mode.

6 Creating an object in the dataset

In the dataset, an object can be used to store static data in a Quaestor knowledgebase. This can be very convenient, especially when large amounts of data need to be stored as input for calculations. An object can be created at any time and it can be filled with data later, either manually or as a result of calculations.

In this paragraph, an object is created that will be given contents later on:

- In the **Knowledge Browser**, select the class *Top Goals/Undefined*. In the right hand side, right-click and select *New Parameter/Function* (or press *Ctrl+I*). Name the Object `Ships` and select the type: *Object*.

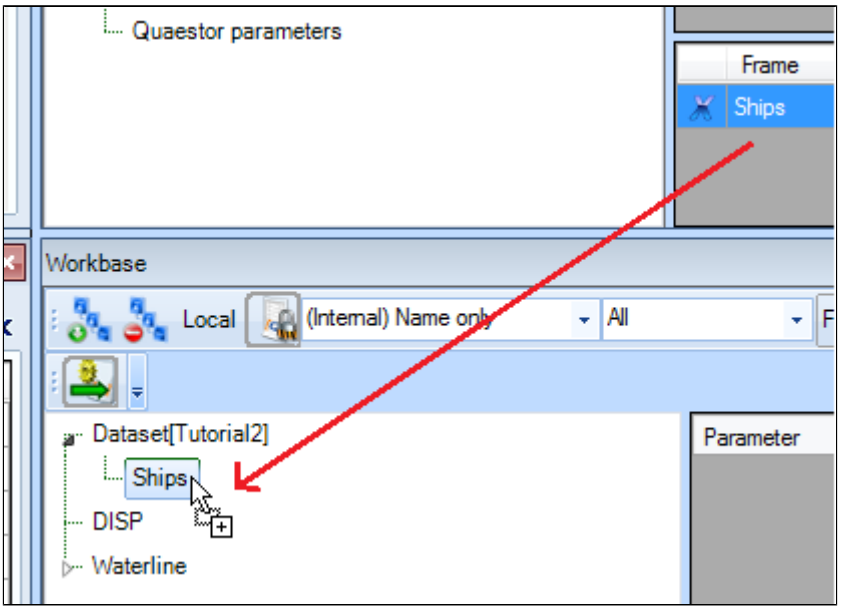
Unlike the `Waterline` object, which was defined by a function, the object `Ships` does not yet meet the validity conditions.

- In the **Properties** window, change the *Determined by* field to *OBJ: Value from Object/Database*.

`Ships` is now a valid object in the knowledge base, but is not yet present in the dataset in the Workbase. You'll have to add it.

- Drag the `Ships` parameter in the **Knowledge Browser** to the `Dataset` node in the **Workbase**.

The object is now added to the dataset, and can be filled with content.

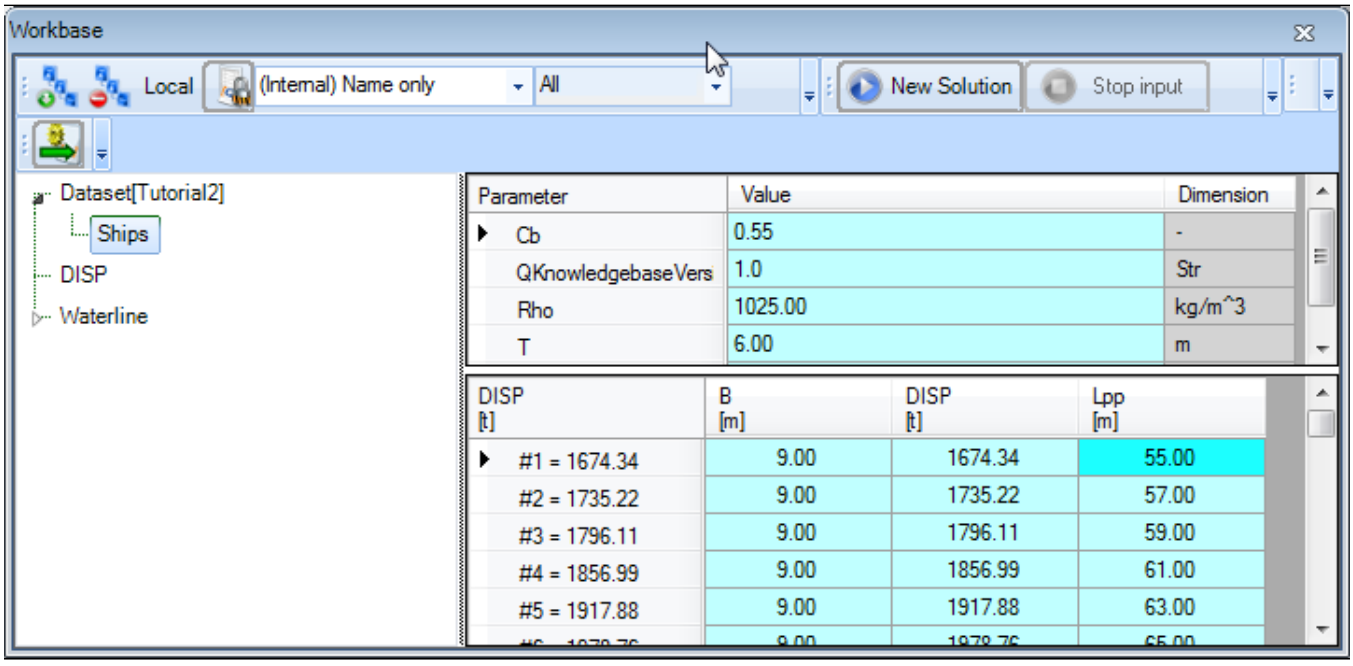


7 Adding content to a data object

Our data object should contain all the ship variations produced in tutorial 1. Therefore, you are going to copy the results of the `DISP` solution (with the variations) to the object `Ships` using the **clipboard**.

- In the Workbase, right-click on the `DISP` solution and select *All to Clipboard* (or press *F4*). The **Workbase Clipboard** now pops up. Click on *Preview*. The contents of the `DISP` solution are now displayed. You temporarily have to use a workaround now, because the Paste function is not yet implemented. Press *Ctrl+A* followed by *Ctrl+C* to copy the contents to the Windows clipboard. Click *Cancel* and close the clipboard. No need to save the values in there. Then right-click on the `Ships` object in the Dataset and select *Database Input* or press *Shift+F3*. In the pop-up window that appears, select *Use Editor* and click *Continue*. In the window that appears next, press *Ctrl+V* to paste the data in the object. Click *OK*. Select *Yes to All* and click *Continue*.

The object `Ships` is now filled with all the variations in length and width:



8 Accessing data in an object

- In the class `Geometry`, create a new relation:

```
B_Frame = 2 * B * Rel_B
```

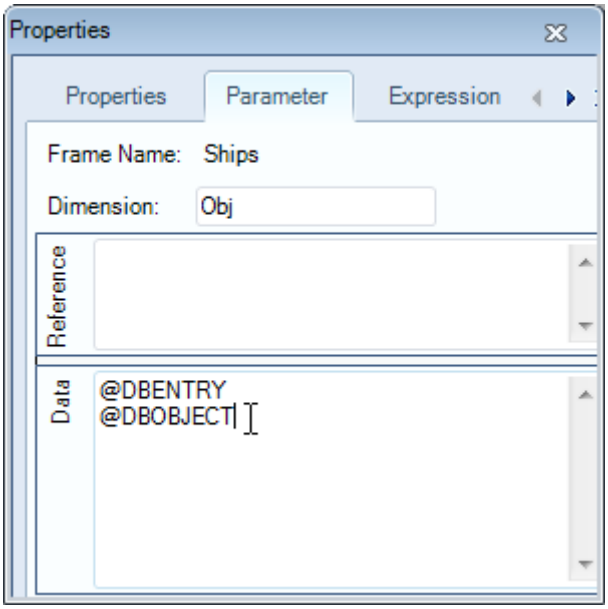
  The dimension of `B_Frame` is *meters (m)*.

This function states that the width of the ship at a certain frame number equals the relative width at that frame number multiplied with the total width of the ship. Notice that the frame number is no direct input in this relation, but Quaestor will still ask for it in a solution as *Rel_B* depends on that parameter.

There is an alternative way of creating solutions. First, you need to add some ***attributes*** to the `Ships` object. Attributes are used to direct the behaviour of Quaestor in solutions. All available attributes are described here. Two attributes are needed: *@DBENTRY* and *@DBOBJECT*.

- Select the object `Ships` in the **Knowledge Browser** (it was created in *Top Goals/Undefined*). In the **Properties** window, select the *Parameter* tab. and in the *Data* field type: `@DBENTRY @DBOBJECT`

Attributes are either separated by a space or by a hard return, not by a comma. The meaning of the entered attributes will be explained at the end of this tutorial.



9 The Process Manager

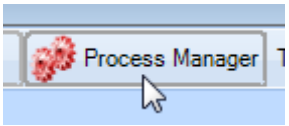
The main procedure to create solutions with an existing dataset is through the use of the **Process Manager**. In the Process Manager, datasets and top goals can be selected to create new solutions. Furthermore, existing solutions, macros or scenarios (these terms will be explained in another tutorial) can be restarted. First, make sure the top goal *`B_Frame`* can be selected in the Process Manager.

- In the Knowledge Browser, *drag* the parameter `B_Frame` to the class *Top Goals/Undefined*.

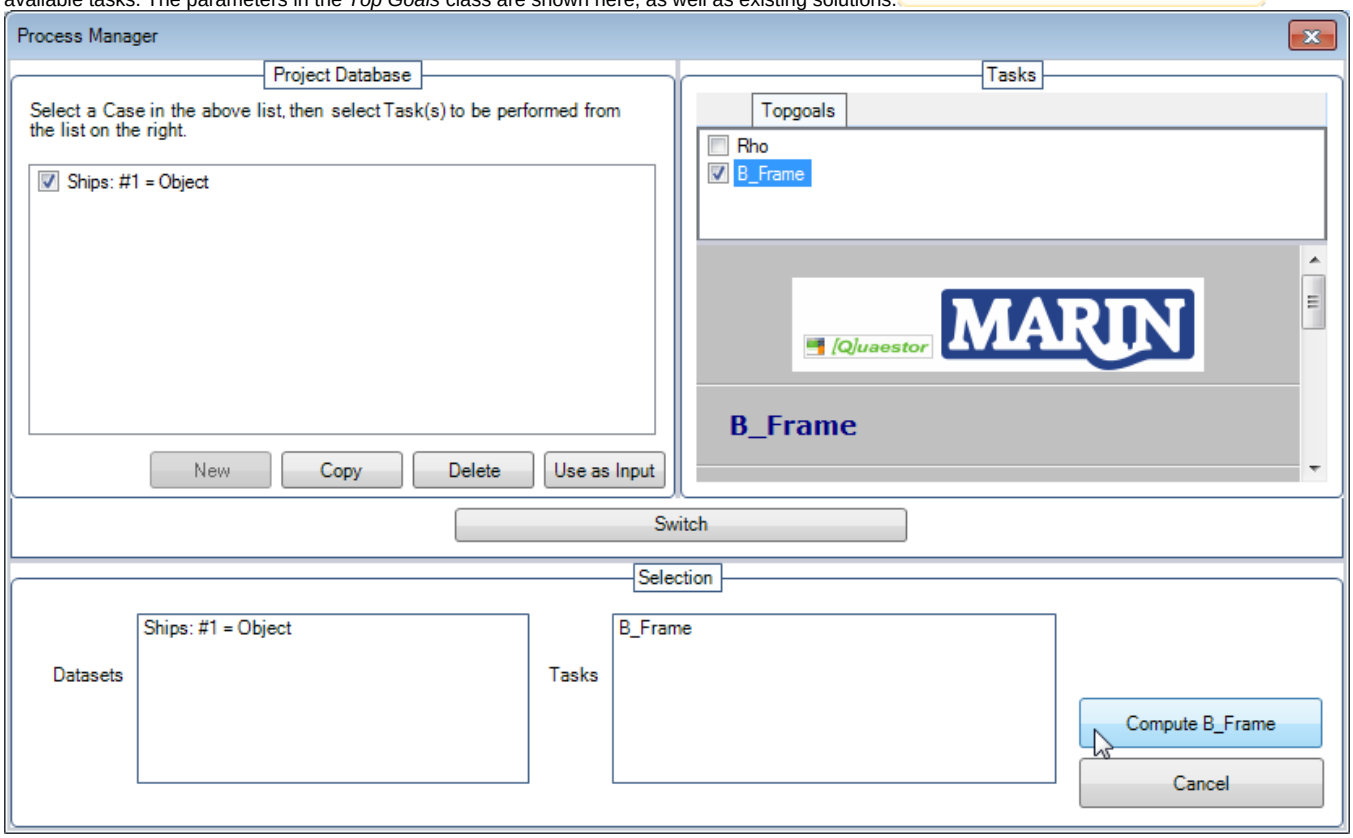
> Parameters can always be exchanged between classes, there is no influence on the behaviour of Quaestor except that parameters in the *Top Goals/Undefined* class are visible in the process manager.

Let's start a solution by means of the process manager.

- In the **Workbase**, click the *S* button or press the "Process Manager" button (Workbase buttons) to start the Process Manager:



The Process Manager starts. The left hand side shows all available datasets, in this case only your `Ships` object. The right hand side shows the available tasks. The parameters in the *Top Goals* class are shown here, as well as existing solutions.



- In the Process Manager, select *Ships:#1* as dataset and *B_Frame* as task. Click the *Compute B_Frame* button. A solution `B_Frame[Ships:#1]` is created.

10 Creating a case matrix

The purpose of all this is to create a table with values of `B_Frame` for every ship in the dataset over the full length (`Frame` 0 to 20). For a first run of a solution, no range use is allowed, except when you specifically tell Quaestor that it is:

- In the **Knowledge Browser**, select the `Top Goals/Undefined` class.
- Select the parameter `B` and in the **Properties** window, select the *Parameter* tab. In the *Data* field, enter: `@RANGEALLOWED`.
- Do the same for the `Frame` parameter.
- In the **Workbase**, for `B`, enter as value `9(0.5)11` and for `Frame`, enter as value: `0(2)20`, then click *Next*. Answer *Yes* to the question about the case matrix.

The result of the calculation is a table with `B_Frame` for every ship of the dataset over the full length (frames 0 to 20).

B [m]	B [m]	B_Frame [m]	Frame [-]	Rel_B [-]
#45 = 11.00	11.00	0.00	0.00	0.00
#46 = 11.00	11.00	3.96	2.00	0.18
#47 = 11.00	11.00	7.04	4.00	0.32
#48 = 11.00	11.00	9.24	6.00	0.42
#49 = 11.00	11.00	10.56	8.00	0.48
#50 = 11.00	11.00	11.00	10.00	0.50
#51 = 11.00	11.00	10.56	12.00	0.48
#52 = 11.00	11.00	9.24	14.00	0.42
#53 = 11.00	11.00	7.04	16.00	0.32
#54 = 11.00	11.00	3.96	18.00	0.18
#55 = 11.00	11.00	0.00	20.00	0.00

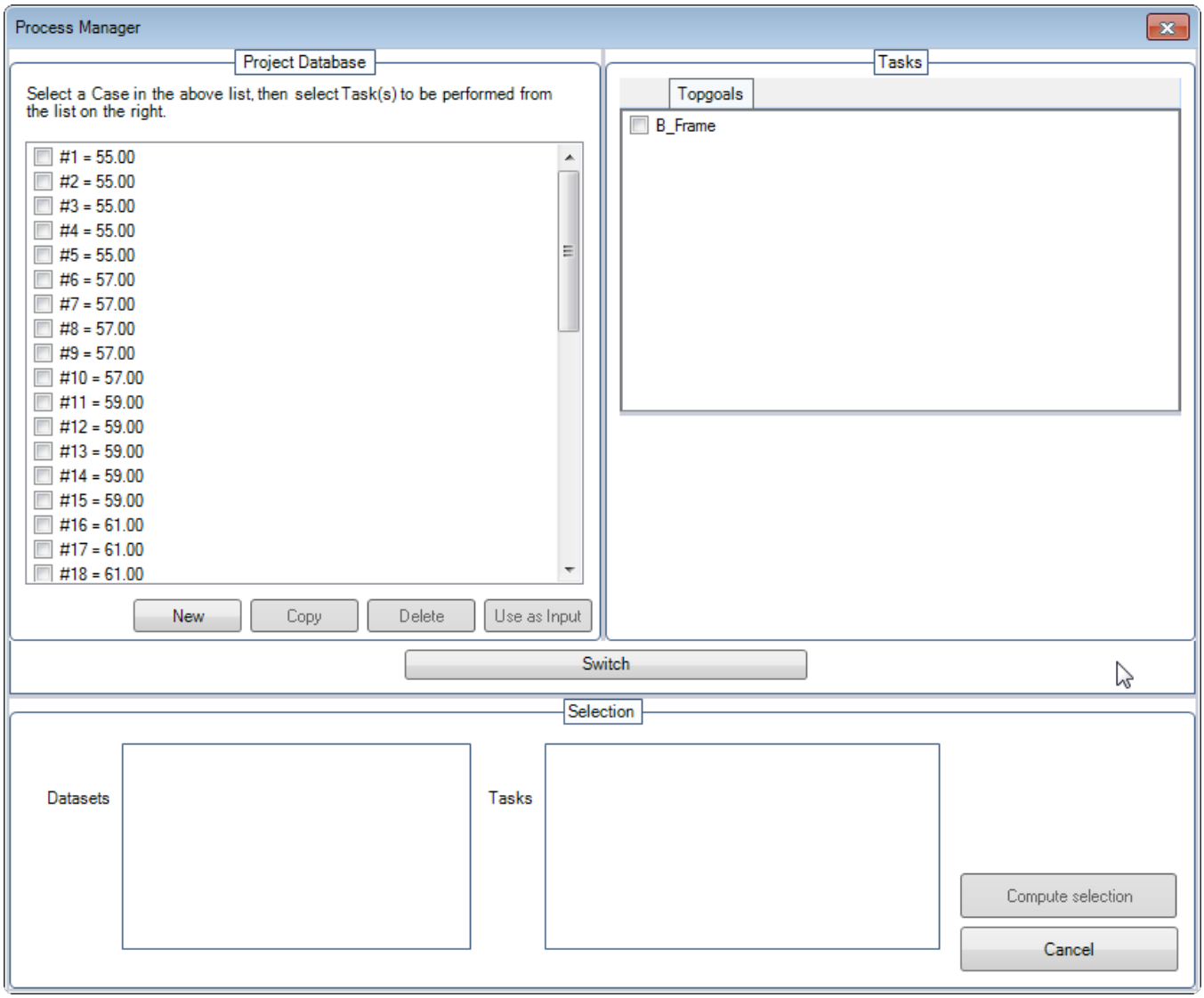
11 @DBENTRY, @DBOBJECT and the dataset

Finally, let's explain the attributes used. The @DBENTRY and @DBOBJECT are used to configure access to the Dataset containing objects. This is best experienced by the difference in behaviour of the process manager.

The dataset of Quaestor can only contain objects on its top level (below the Dataset node). When you want to use the data in the Dataset, you can either select the object or define the object as database entry so that the Process manager will show it's content. @DBENTRY defines this entry point for selection of data by Quaestor (and thus in the process manager).

@DBOBJECT is an addition to the entry point definition and defines that an object may be available in the Dataset as multi case value. For instance several ships.

Try running the process manager without @DBOBJECT. It's now possible to select the cases within the Ships object, instead of the whole object (all cases). The reason is that Quaestor now expects there is only one relevant object, being the one specified as @DBENTRY, so only selections within this object are relevant to be shown in the process manager.



Please note that @DBOBJECT is an addition to @DBENTRY. **So @DBENTRY should always be in the multi case object you want to use as entry point for your data in combination with @DBOBJECT**.
Furthermore, at this moment it is not possible to do calculations for several multi case objects at once (this behaviour is suggested in earlier Quaestor versions in situations where you use @DBOBJECT without @DBENTRY and as a result are able to select several or All Ships...).

12 Check

You can verify your results by comparing it to [Tutorial 2 Finished]

<< Back to tutorial 1 – Continue with tutorial 3 >>