Simulation software for safe, sustainable and future deltas

Delft3D FM Suite

Deltares systems

Delta Shell

User Manual

Deltares

Delta Shell

User Manual

Released for:
Delft3D FM Suite 2024
D-HYDRO Suite 2024
SOBEK Suite 3.7

Version: 1.2
Revision: 78359

16 July 2024

Delta Shell, User Manual

Published and printed by:
Deltares
Boussinesqweg 1
2629 HV Delft
P.O. 177
2600 MH Delft
The Netherlands

telephone: +31 88 335 82 73
e-mail: Information
www: Deltares

For sales contact:
telephone: +31 88 335 81 88
e-mail: Sales
www: Sales & Support

For support contact:
telephone: +31 88 335 81 00
e-mail: Support
www: Sales & Support

Contents

List of Tables

List of Figures

1 A guide to this manual

1.1 Introduction

This User Manual concerns the Delta Shell framework.

This module is part of several Modelling suites, released by Deltares as Deltares Systems or Dutch Delta Systems. These modelling suites are build with use of the Delta Shell framework. The framework enables to develop a range of modeling suites, each distinguished by the components and — most significantly — the (numerical) modules, which are plugged in. The modules which are compliant with the Delta Shell framework are released as D-*Name of the module*, for example: D-Flow Flexible Mesh, D-Waves, D-Water Quality, D-Real Time Control and D-Rainfall Runoff.

Therefore, this User Manual is shipped with several modelling suites. On the *Start Page* links are provided to all relevant User Manuals (and Technical Reference Manuals) for that modelling suite. Other user manuals can be referenced. In that case, you need to open the specific user manual from the *Start Page* in the central window. Some texts are shared in different user manuals, in order to improve the readability.

1.2 Overview

To make this manual more accessible we will briefly describe the contents of each chapter.

Chapter 2: Background, introduces the Delta Shell framework as an integrated modelling environment to provide users with a single application which acts as a platform to integrate various models and tools.

Chapter 3: General overview of the GUI, gives a brief introduction to all GUI-components, which are shared between applications based on the Delta Shell framework.

Chapter 4: Case management, introduces two ways to work with cases, (model) variants or scenarios.

The map is always a very - if not, the most - usefull window in modelling in Delta Shell. Read more about the map and coordinate systems in Chapter 5: Working with the map.

Chapter 6: Spatial editor, introduces the spatial editor, a generic feature of the Delta Shell for editing spatial data, such as bathymetry, roughness, viscosity, initial conditions, sediment availability.

Chapter 7: Model coupling, is on models, (timestep based) model coupling and so-called integrated modelling within Delta Shell.

Chapter 8: Command-line and scripting, describes how to run a model without using the GUI. For the advanced user, there is also an introduction to the very powerfull feature of scripting, which enables the user to adjust and run a model without using the GUI, even to setup complete models from scratch.

1.3 Manual version and revisions

This manual applies to SOBEK 3 suite (version 3.5 and higher) and Delft3D Flexible Mesh Suite (version 2016 and higher).

1.4 Typographical conventions

Throughout this manual, the following conventions help you to distinguish between different elements of text.

Example	Description
Module **Project**	Title of a window or a sub-window are in given in **bold**. Sub-windows are displayed in the **Module** window and cannot be moved. Windows can be moved independently from the **Module** window, such as the **Visualisation Area** window.
Save	Item from a menu, title of a push button or the name of a user interface input field. Upon selecting this item (click or in some cases double click with the left mouse button on it) a related action will be executed; in most cases it will result in displaying some other (sub-)window. In case of an input field you are supposed to enter input data of the required format and in the required domain.
<\tutorial\wave\swan-curvi> <siu.mdw>	Directory names, filenames, and path names are expressed between angle brackets, <>. For Linux environments a forward slash (/) is used instead of the backward slash (\) for Windows environments.
"27 08 1999"	Data to be typed by you into the input fields are displayed between double quotes. Selections of menu items, option boxes etc. are described as such: for instance 'select *Save* and go to the next window'.
`delft3d-menu`	Commands to be typed by you are given in the font Courier New, 10 points.
➢	In this User manual, user actions are indicated with this arrow.
[m s^{-1}] [–]	Units are given between square brackets when used next to the formulae. Leaving them out might result in misinterpretation. Most units will be in SI notation. [m AD] stands for 'meter Above Datum', which denotes a level relative to the vertical reference system in the model.

1.5 Changes with respect to previous versions

In this edition changes have been made to:

- the chapter 6: Spatial editor
- the Appendix A: How to use OpenDA for Delta Shell models

2 Background

2.1 About the Delta Shell framework

After its first release in 2012, the Delta Shell framework will gradualy be extended to become a full-fledged framework for the integration of a variety of environmental models.

The Delta Shell framework is an integrated modelling environment to provide users with a single application which acts as a platform to integrate various models and tools. This is achieved by making use of a software framework specifically focused to provide a set of components which can be reused by all kinds of environmental models. The overall picture of what functionality Delta Shell needs to provide has been proposed by a variety of (potential) users and has been captured in so-called user stories. These user stories have been gathered in a so-called mind map which is presented in Figure 2.1. As can be seen from the mind map,

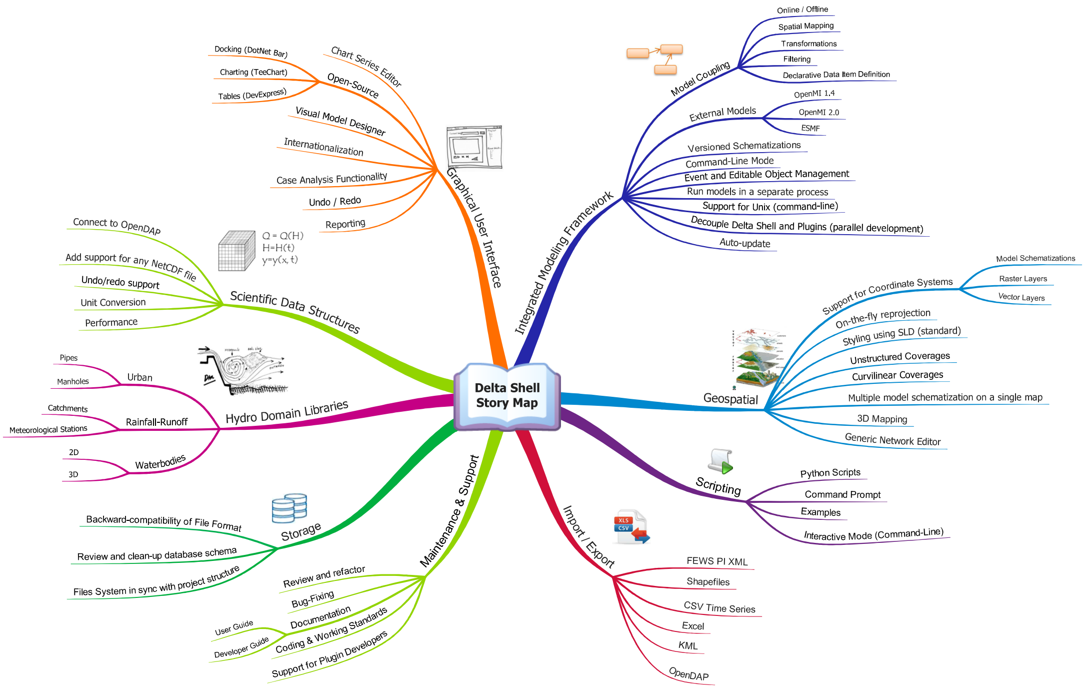


Figure 2.1: *Mindmap with key user stories within Delta Shell.*

Delta Shell has to comply with a broad range of wishes defined by the various users and user groups. Main topics (depicted in the main branches) are:

- ◇ The definition of the integrated modelling framework,
- ◇ Data management: structuring, importing/exporting, storage.
- ◇ Dealing with environmental domain libraries (hydro, geo).
- ◇ Application interfaces (APIs) such as a graphical user interface or a command line interfce using scripts for batch operation.
- ◇ Documentation and support

Delta Shell provides a user-friendly and open framework for environmental applications. Figure 2.2 shows the principles of the Delta Shell framework. Delta Shell allows to combine different modules (plug-ins), such as for example, D-Flow 1D, D-RTC, D-Water Quality, or D-RR, which results in the SOBEK 3 suite. In this fashion, it is possible to compose various dedicated software suites within a single framework and preserving the same look-and-feel at the same time. Furthermore, Delta Shell contains common plug-ins with generic functionality which may be used by all model plug-ins that have been integrated within the framework. Finally, Delta Shell contains tools for setting up or importing different types of models, perform simulations of the different models or combinations of models and analyse model results.

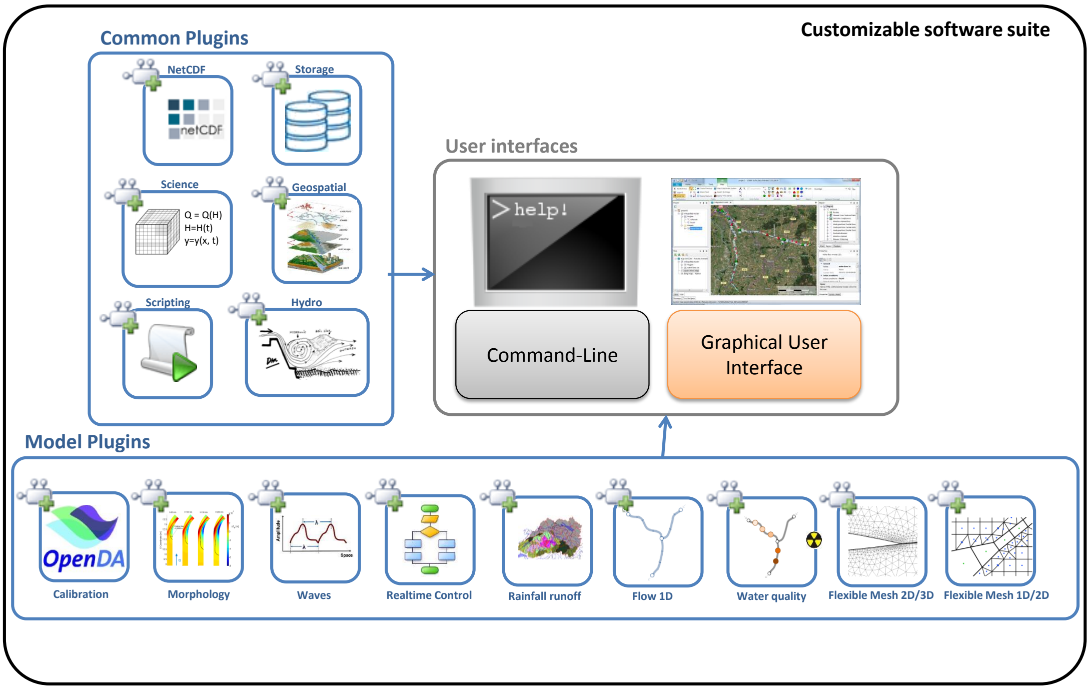


Figure 2.2: *The structure of Delta Shell as integrated modelling suite.*

A model is by definition a simplified version of (a possible) reality. A generic model is a mathematical description of physical processes provided as a computer program, for example D-Flow 1D, which is a model of surface water flow that solves the Saint-Venant equations in one dimension. D-Flow 1D is a numerical model, because the governing flow equations are solved numerically. In combination with data for a specific part of the world the model becomes a site-specific model, for example the D-Flow 1D-model for the river Rhine from Maxau to Lobith. Within a site-specific model a set of boundary conditions forms a scenario.

Delta Shell follows a layered concept to get from 'real world' to 'numerical model result' (Figure 2.3). Real objects like rivers, lakes, buildings, hills, bridges, pumps, culverts, etc. usually are available on maps or other (digital) format. Based on such data Delta Shell helps to create a schematisation, i.e. a network with model objects that correspond to the real objects. The schematisation makes the model site-specific. A computational grid is added, which is part of the discretisation of real world processes like surface water flow into a numerical model. This model can be run under different sets of boundary conditions (scenarios). Numerical solution of the flow equations with a Delta Shell plugin finally produces model results.

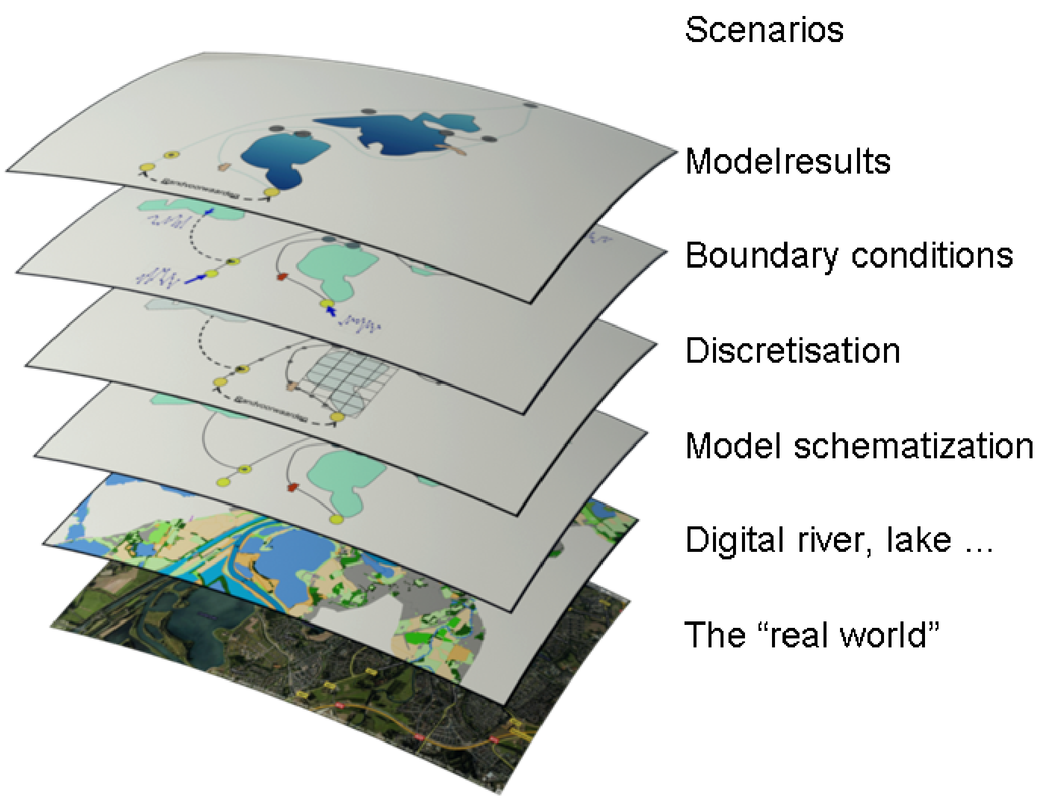


Figure 2.3: *From real world to model results with Delta Shell.*

2.2 Delta Shell: Vision Statement

For users:
The framework should facilitate modellers in: schematization (with/without user interface), calibration of models, running of models in standardized modelling environments (ex. OpenMI), batch execution (from command line) and parallel execution (on clusters). All modellers should be able to use the Delta Shell Framework combined with the modelling tools of their preference (ex. D-Flow 1D/2D, D-Flow FM, Simona, Mike11, etc.).

For developers:
The Delta Shell Framework should provide a generic, transparent and simple way of developing new or coupling existing modelling tools (plug-ins). The emphasis should be on reuse and expansion of existing code and features.

2.3 Overall Architecture

This section gives an overview of the Delta Shell architecture and a description of its main components. Delta Shell has been designed using a flexible architecture which can easily facilitate the use of external applications. The concept of the overall architecture is similar to existing commercial products like Eclipse or Microsoft Visual Studio Shell. The most important subjects considering the architecture of Delta Shell are presented in Figure 2.4. These are the subjects that need to come together within the architecture of Delta Shell.

The subjects within the main branches of the mindmap that grasp the overall architecture of Delta Shell are:

◇ Interoperability and/or Model Coupling. Define an architecture that is able to couple a

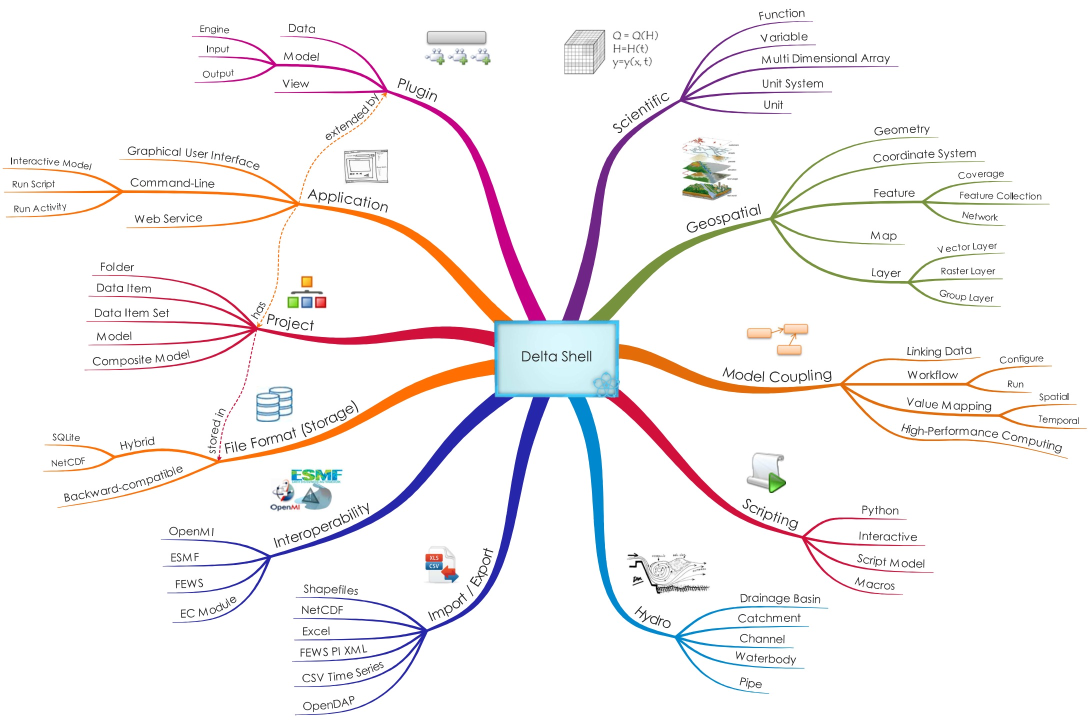


Figure 2.4: *Mindmap with key architecture subjects of Delta Shell.*

variety of models in a correct fashion, both numerical as physics-wise.

- ◇ Applications and plug-ins. Provide a flexible framework architecture such that it becomes possible to build applications (such as a command-line runner) or integrate (model) plug-ins within the framework in a generic fashion.
- ◇ Data. The Delta Shell framework needs to provide functionality to deal with the storage, import/export, transformation of data, as well as to provide support for the scientific character of (most of) these data.
- ◇ Domains. Currently, Delta Shell needs to provide functionality for the Hydro and Geospatial domains. Support for other domains is foreseen in the future.
- ◇ Scripting. To be able to set-up a Delta Shell project from the command-line, or perform batch simulations using Delta Shell, it needs to support scripting.

To grasp all of the above subjects within the Delta Shell framework, the architecture of Delta Shell consists of multiple layers that will all be described in this document. The resulting overall architecture of Delta Shell is depicted in Figure 2.5.

The highest level of the Delta Shell architecture is shown in Figure 2.5. The central component of the system is the modeling framework. The framework consists of a set of class libraries which can be used throughout the system by the common and application plug-ins and stand-alone applications such as the Graphical User Interface (GUI) or command-line runner.

To integrate new plug-ins within Delta Shell the framework provides interfaces for the plug-in to comply with via its Core and GUI APIs (for example the IPlugin, IModel, IModelProvider, IView or IViewProvider interfaces). The same holds for adding stand alone top level applications (using the IApplication interface). Currently Delta Shell offers 2 of these applications, e.g. the Delta Shell GUI and the Delta Shell command line runner, both of which combine the available plug-ins within a single environment. It should be noted that other extensions of the Delta Shell environment, for example an ArcGIS Extension, or Web Service, can easily be developed as an alternative to existing GUI and console applications. Only the top-level part

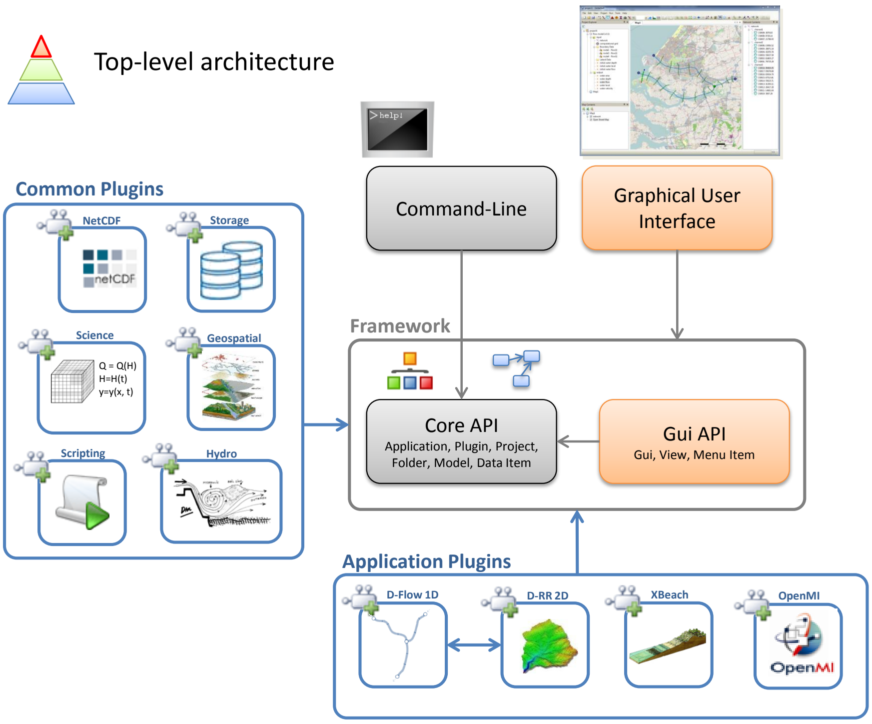


Figure 2.5: *Top-level architecture of Delta Shell.*

of the system needs to be replaced and the rest of the components will work as before without making any changes.

The plug-ins have been implemented as extensions to Delta Shell and can be divided into two groups. The first group consists of the so called common plug-ins which provide common functionality such as data types (network data types, OGC-compliant GIS data types, multi-dimensional data types etc.) or which provide generic functionality such as the import/export of data, the editing of data using various custom controls, or the visualization of data. The functionality implemented in these common plug-ins is made available to the second group of application plug-ins implicitly via the framework, and explicitly by direct use of .NET assemblies such as the GeoAPI or NetCDF assemblies (dll's). This second group consists of specific plug-ins which integrate some computational model (currently D-Flow 1D, D-Water Quality, D-RTC, and D-RR) into the system.

3 General overview of the GUI

As shown in Figure 2.5, one of the interfaces within the User Interface makes it possible to extend the available (model) plug-ins with Graphical User Interfaces (GUIs). In this fashion, all user interfaces of model plug-ins integrated within Delta Shell have the same look-and-feel.

This chapter introduces all GUI-components, which are shared between applications based on the Delta Shell framework.

3.1 Windows

As Delta Shell is an integrated modelling suite, the application is project-based. Within a project several models may be run and combined.

The main user interface is organized in a set of tool and document windows. An example is given in Figure 3.1. The tool windows show properties of the current project, whereas document windows are used to visualize or edit a specific data type. Tool windows can be docked where you prefer — even at a second display. Document windows are, when placed within the framework, always in the central area but may also be docked stand-alone (on a second display, for example). Examples of tool windows are:

◇ **Project**
◇ **Map**
◇ **Region**
◇ **Properties**
◇ **Chart**
◇ **Data**
◇ **Time navigator**
◇ **Messages**
◇ **Toolbox**

Examples of document windows are:

◇ Map(s)
◇ Editor(s)

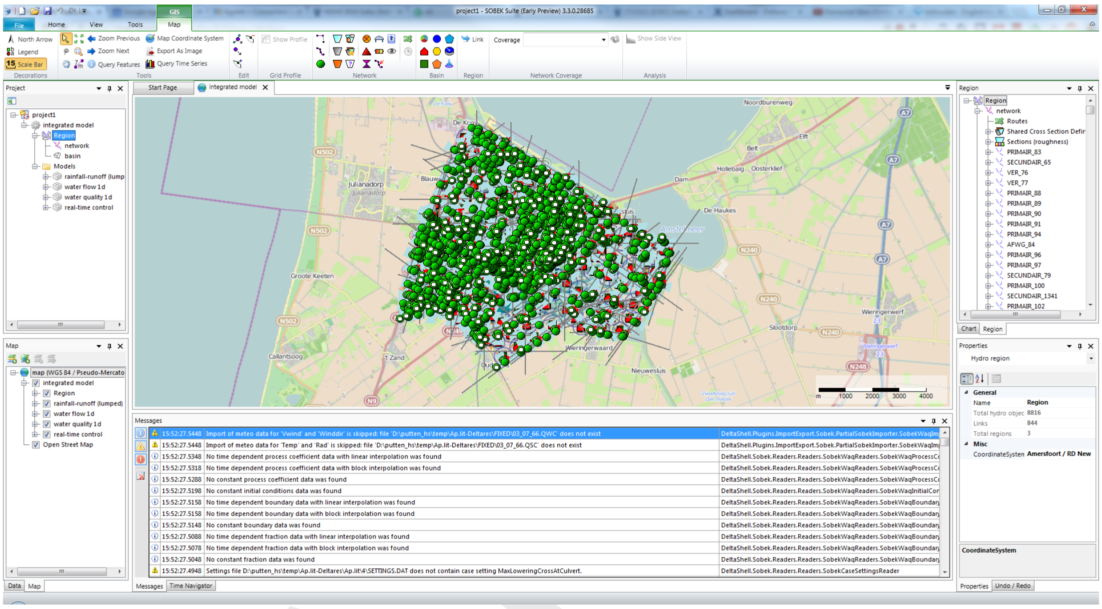


Figure 3.1: Overview of the graphical user interface, example for a SOBEK 3 model.

In this chapter all tool windows, menus, dockable views, context menus, and ribbons and toolbars will be described. Map functionality and the spatial editor are treated in separate chapters. The specific editors for the different models are described in the user manuals belonging to those model plug-ins.

3.1.1 Project

The **Project** window is the main navigation window for the project data, showing the total workspace in a tree view (Figure 3.2). In the **Project** all project components are shown. For a project containing a one-dimensional flow model and a map these may look like:

- The model (in this case a 1-dimensional flow model)
  - The model input
    - The network(s)
    - The computational grid(s)
    - The initial conditions
    - The boundary conditions
    - Lateral data
    - Roughness
    - Wind data
  - The model output/results
- The map

All project items with sub-levels can be collapsed by a mouse-click on the '—' sign in the tree view. Project data can be sorted by adding new folders to the project tree view and moving models or movable items to designated folders. By clicking on the top left icon in Figure 3.2 the active item in the central Map is located in the tree view.

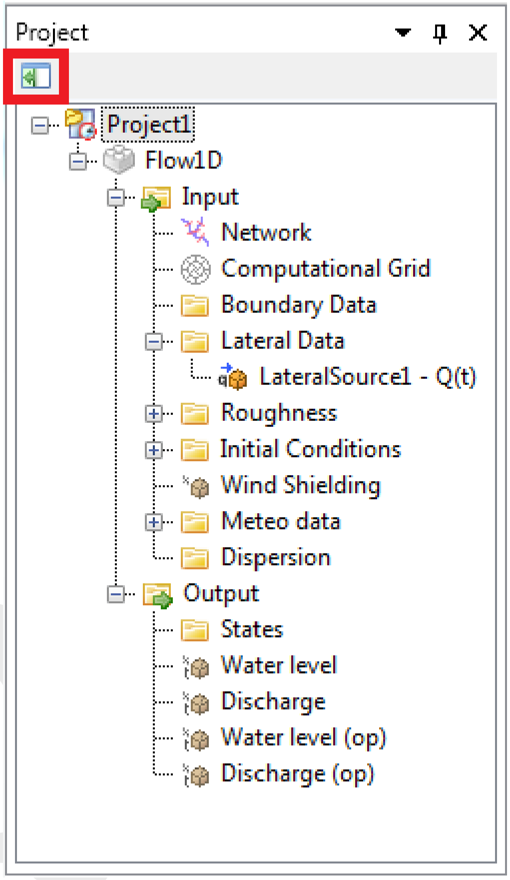


***Figure 3.2:** The project tree window.*

Several possibilities exist to work with the tree view:

- ◇ Left mouse-click to select
- ◇ Right mouse-click gives a context menu with available actions
- ◇ Double-click to show a map or editor in the main (central) window, depending on the parameter

3.1.2 Main (central) window

The main window (Figure 3.3) is by default always placed in the middle of the screen. It can also be docked separately, for example on a second display. It is used to present a map for all geo-referenced modeldata, the editors for other data, and results in charts. The editors for other data are model-specific and therefore described in the manuals for the various model plug-ins.

All items with a geo-reference will be presented on the central map, for example: network, computational grid and output data as layers, comparable to a geographical information system (GIS). Working with these layers is described in section 5.2.

Figure 3.3: *The central map view.*

When working in the central map, for example on a network, it is possible to add, adjust or delete network components, which is described thoroughly in the D-Flow 1D manual.

Results in charts windows (Figure 3.6) are presented by combining a table view and a 'xy'-plot. The user can visualize separate data points by selecting rows in the table view. The data that is presented in the chart window may also be exported to a *.csv file by clicking the *Csv export* button.

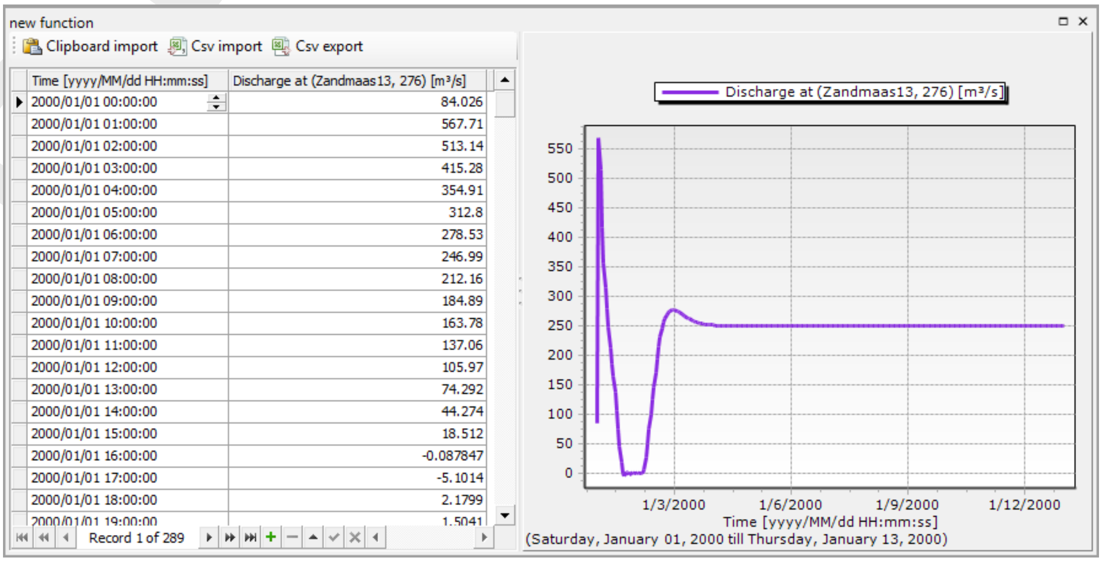


Figure 3.4: *The chart window view.*

3.1.3 Map

The **Map** window (Figure 3.5) manages the active map. In this window layers within the active map can be shown, hidden or adjusted. With the four icons in the top left of the window new <shp>- or <wms>-layers can be added, removed or exported. With the icons in the top right of the window, the window can be removed or hidden. The window can be retrieved by clicking on *Map* in *View* ribbon.

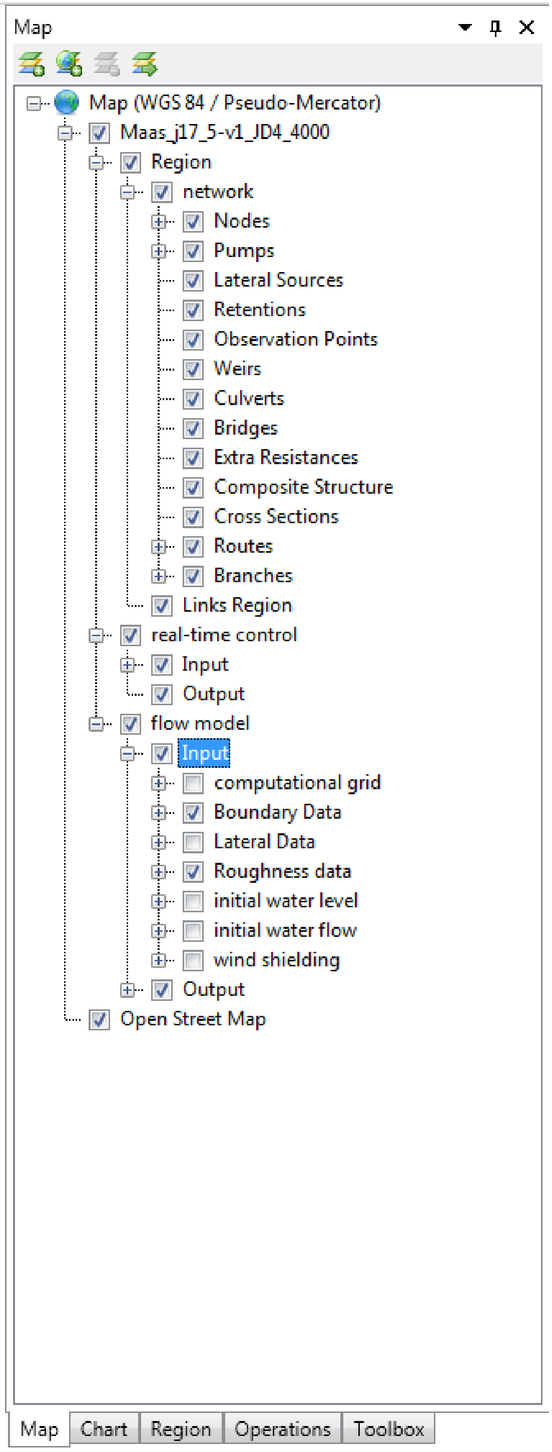


Figure 3.5: *The map window.*

3.1.4 Chart

The **Chart** window (Figure 3.6) can be used to stack/unstack chart series of the same type or to convert chart series to a certain type (area, line, point, or bar series), Furthermore, series can be selected or deselected within this window.

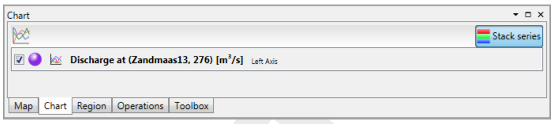


Figure 3.6: *Example of the chart window.*

3.1.5 Region

In the **Region** window (Figure 3.7) all hydrological objects are shown, like river-networks, -catchments or waterbodies. The **Region** window is related to the selected model. While in the central map the model objects are arranged according to their spatial location, the **Region** sorts the model objects by category and chainage. A component that is selected in the **Region** window is also automatically selected in the central map and vice versa.

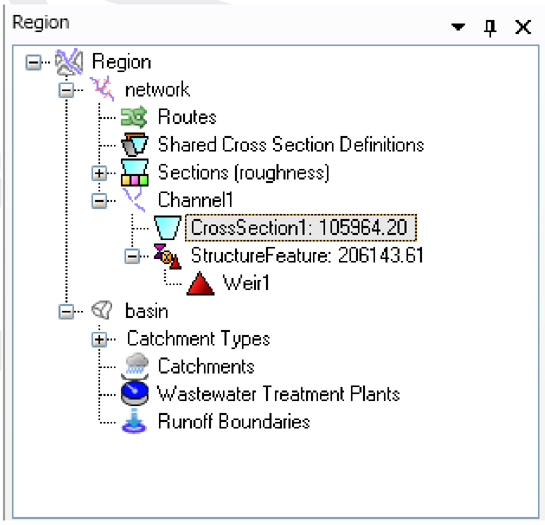


Figure 3.7: *Example of the region window.*

3.1.6 Toolbox

In the **Toolbox** window (Figure 3.8), users can quickly add a variety of items (models, stand-alone items) to the project tree, by selecting 'Add new item' or 'Add new model' after a right-mouse-click on the desired item. Furthermore, users may manage their Python scripts within the **Toolbox** window.

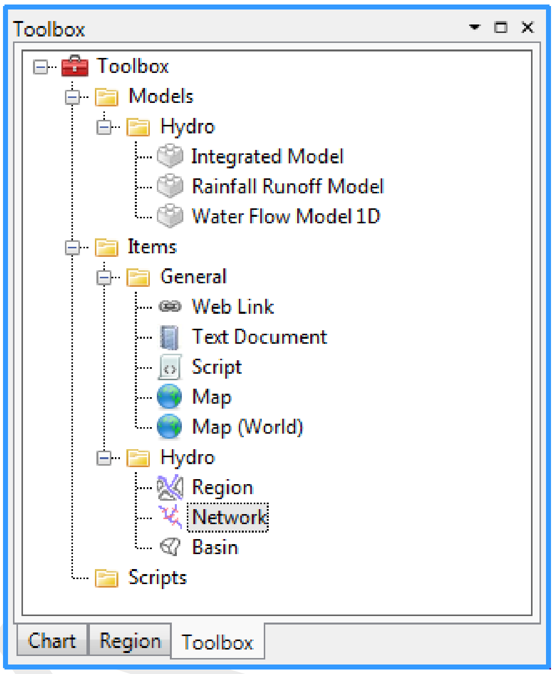


Figure 3.8: *Example of the **Toolbox** window.*

3.1.7 Properties

The **Properties** window shows properties for an active selection of the graphical user interface. When a model object is selected in the **Region** window it shows the properties of this object. Accordingly, the **Properties** window of an item selected in the **Project** shows data related to the selected item, for example the simulation time of a <flow model> or a list with output parameters when clicking on the <output> entry. Figure 3.9 shows an example for the properties of a flow model.

In the **Properties** window data can also be edited. If the property grid is insufficient to display the information, for example in case of time series, an additional editor can be opened.

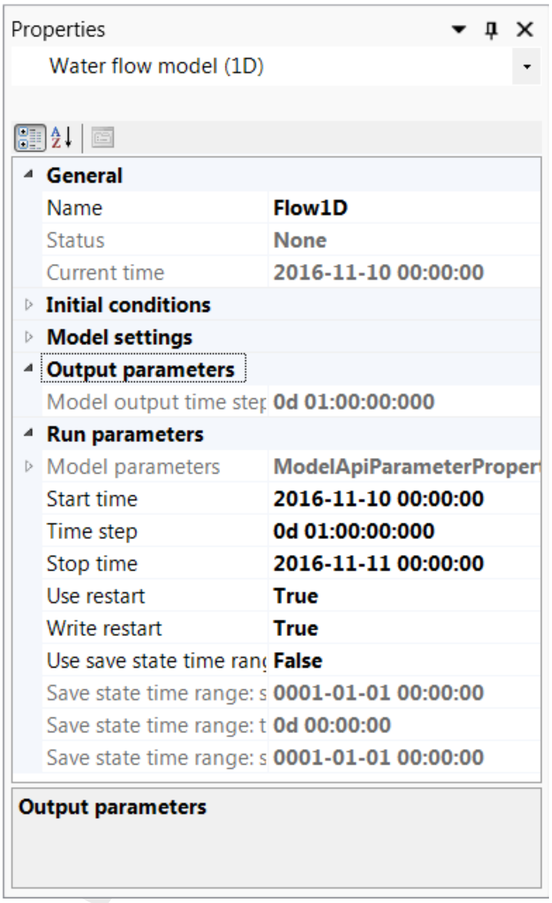


Figure 3.9: *Example of a property grid in the properties window of a flow model.*

3.1.8 Time navigator

The **Time navigator** is used to navigate through time(steps) of any time-dependent variable. An example is given in Figure 3.10. Each map or side view that shows time-dependent variables has its own time navigator. Users may also play results of time-dependent variables continuously.

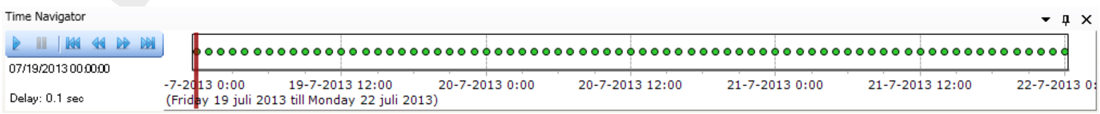


Figure 3.10: *Example of the **Time navigator** window.*

3.1.9 Messages

The **Messages** window (Figure 3.11) is a logging window. Messages sent from models or different parts of the system are shown here. When a message is too large to fit within the **Messages** window, the user can open a single message (**??**) separately by right-mouse-clicking the message and selecting the 'Show details' option.

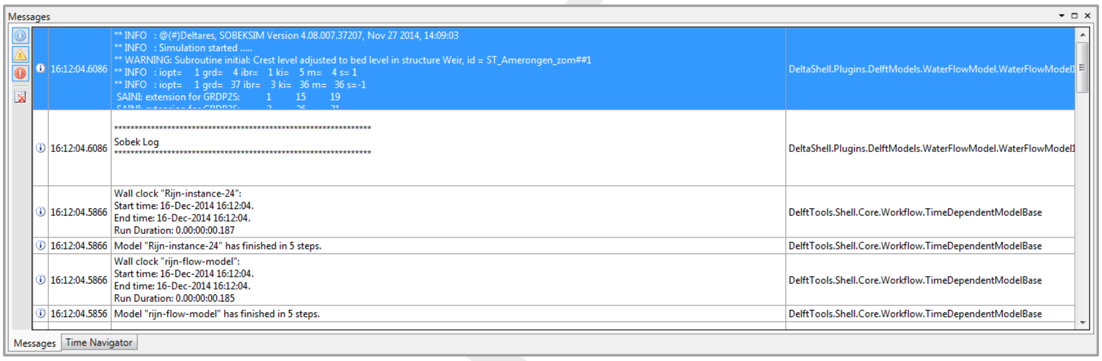


Figure 3.11: *The messages window.*

Within the **Messages** window the user may select the verbosity of the shown messages, ranging from 'Info messages' to 'Warning messages' to 'Error messages'. It is also possible to clear all messages by clicking .

Furthermore, a run report is shown in the output in the **Project** for each model simulation. This run report contains all the messages (from Delta Shell and the model plug-ins) that occur during a simulation.

Finally, an application log is kept for each session of Delta Shell in the project database. In this log-file, which can be accessed through the *File/Help* or *Home* menus, all messages are stored.

3.2 Dockable views

The User Interface offers lots of freedom to customize dockable views, which are discussed in this section.

3.2.1 Docking tabs separately

Within the User Interface the user can dock the separate windows according to personal preferences. These preferences are then saved for future use of the framework. An example of such preferences is presented in Figure 3.12, where windows have been docked on two screens.

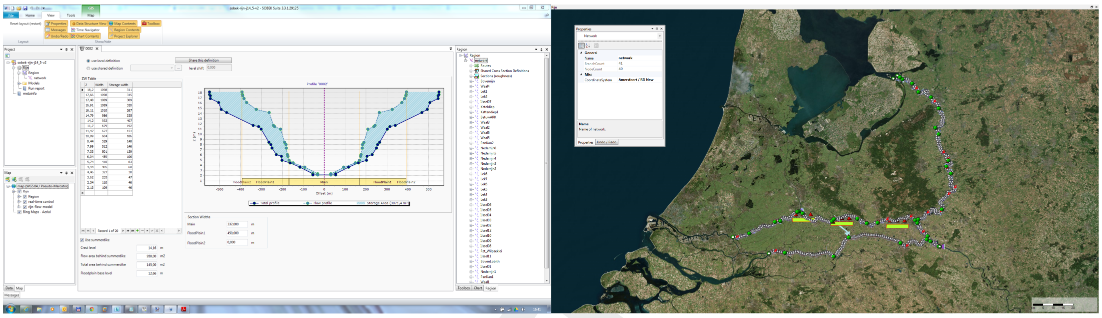


***Figure 3.12:** Docking windows on two screens within the User Interface.*

3.2.2 Multiple tabs

In case two windows are docked in one view, the underlying window (tab) can be brought to the front by simply selecting the tab, as is shown here.

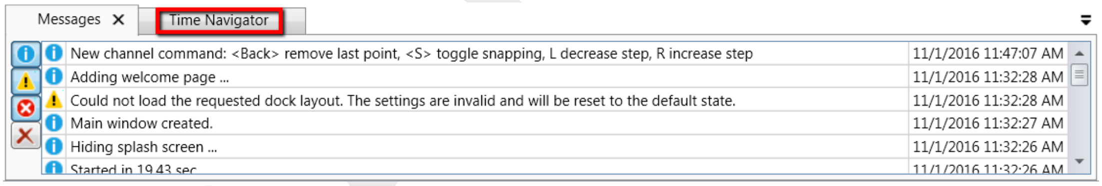


***Figure 3.13:** Bringing the **Time Navigator** window to the front*

By dragging dockable windows with the left mouse button and dropping the window left, right, above or below another one the graphical user interface can be customized according to personal preferences. Here an example of the **Time Navigator** window being docked to the right of the **Messages** window.

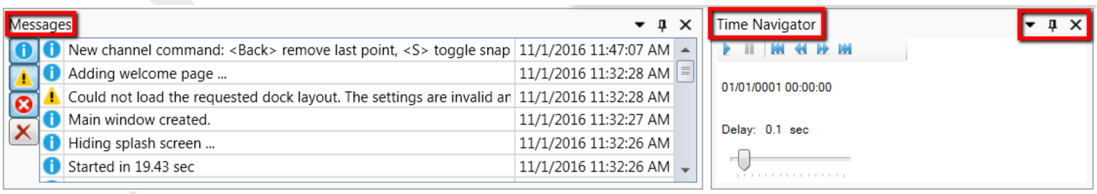


***Figure 3.14:** Docking the **Time Navigator** window.*

Additional features are the possibility to remove or (auto) hide the window (top right in Figure 3.14). In case of removal, the window can be retrieved by two left mouse-clicks on *Time Navigator* in the *View* ribbon. Hiding the **Properties** window results in:

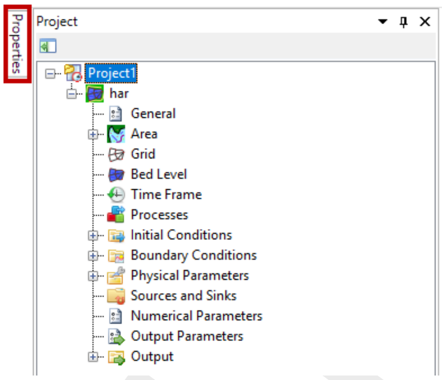


Figure 3.15: *Auto hide the* ***Properties*** *window*

3.3 Context menus

Depending on the active window, different context menus are present when right-mouse-clicking on items within this window. This section will treat these context menus per active window.

3.3.1 Project

Within the **Project** window, a variety of levels and/or items with different context menus are present. These will be described here.

3.3.1.1 Project level

The context menu of the project level within the project explorer is shown in Figure 3.16. It contains the following choices:

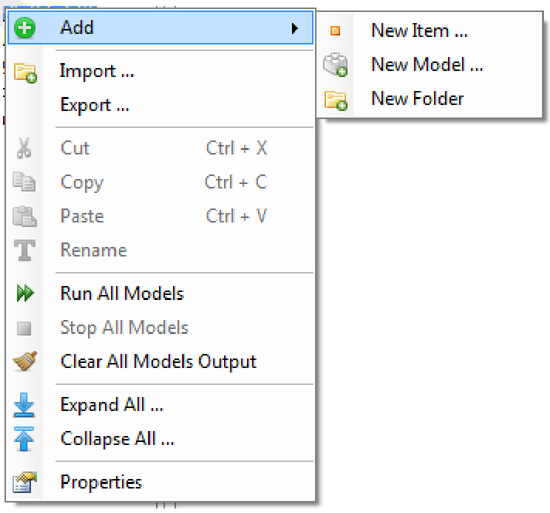


Figure 3.16: *The context menu on the project level within the project explorer.*

◇ *Add* ⇒ Option to add models and/or items to project (depending on installed model plug-ins)
◇ *Import . . .* ⇒ Opens selection window for a variety of available importers (if present)
◇ *Export . . .* ⇒ Opens selection window for a variety of available exporters (if present)
◇ *Cut* ⇒ Cuts current project for pasting elsewhere
◇ *Copy* ⇒ Copies current project for pasting elsewhere
◇ *Paste* ⇒ Pastes current project available on the clipboard

- *Rename* ⇒ Rename the current project
- *Run All Models* ⇒ Runs all models available in the project
- *Stop All Models* ⇒ Stops running of all models currently running within the project
- *Clear All Models Output* ⇒ Clears all model output of models available within the project
- *Expand All ...* ⇒ Expands all project items
- *Collapse All ...* ⇒ Collapses all project items
- *Properties* ⇒ Switches to **Properties** window of active project

3.3.1.2 Integrated model level

The context menu of the integrated model level within the project explorer is shown in Figure 3.17. It contains the following choices:

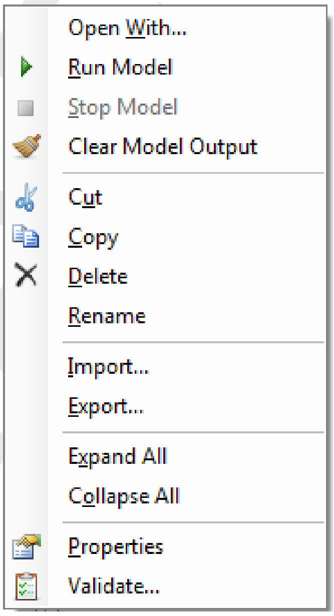


***Figure 3.17:** The context menu on the integrated model level within the project explorer.*

- *Open With* ⇒ Opens a dialog with choices to open the integrated model (for example with 'Hydro Model Settings')
- *Run Model* ⇒ Runs the integrated model (including all its sub models)
- *Stop Model* ⇒ Stops running of the integrated model
- *Clear Model Output* ⇒ Clears the model output of the integrated model (including all its sub models)
- *Cut* ⇒ Cuts the integrated model for pasting elsewhere
- *Copy* ⇒ Copies the integrated model for pasting elsewhere
- *Delete* ⇒ Deletes the integrated model from the project
- *Rename* ⇒ Rename the integrated model
- *Import ...* ⇒ Opens selection window for a variety of available importers (if present)
- *Export ...* ⇒ Opens selection window for a variety of available exporters (if present)
- *Expand All ...* ⇒ Expands all integrated model items
- *Collapse All ...* ⇒ Collapses all integrated model items
- *Properties* ⇒ Switches to **Properties** window of active integrated model

3.3.1.3 Region level

The context menu of the region level within the project explorer is shown in Figure 3.18. It contains the following choices:

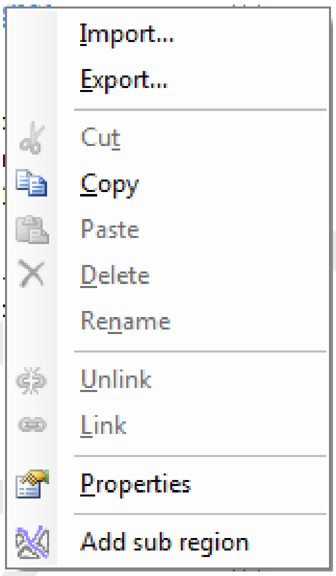


Figure 3.18: *The context menu on the region level within the project explorer.*

- *Import . . .* ⇒ Opens selection window for a variety of available importers (if present)
- *Export . . .* ⇒ Opens selection window for a variety of available exporters (if present)
- *Cut* ⇒ Cuts the region for pasting elsewhere
- *Copy* ⇒ Copies the region for pasting elsewhere
- *Paste* ⇒ Pastes current region available on the clipboard
- *Delete* ⇒ Deletes the region model from the project
- *Rename* ⇒ Rename the region
- *Unlink* ⇒ Unlink region from model
- *Link* ⇒ Link the region to model
- *Properties* ⇒ Switches to **Properties** window of active integrated model
- *Add sub region* ⇒ Opens selection window to add sub regions to region

3.3.1.4 Model level

The context menu of the model level within the project explorer is shown in Figure 3.19. It always contains the following choices:

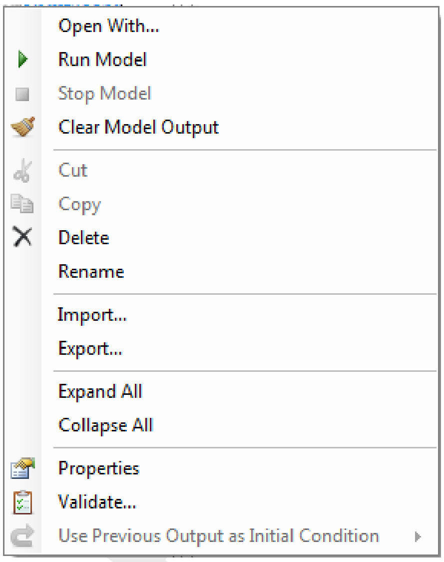


Figure 3.19: *The context menu on the model level within the project explorer.*

- *Open With* ⇒ Opens a dialog with choices to open the model (for example with 'Validation report')
- *Run Model* ⇒ Runs the model
- *Stop Model* ⇒ Stops running of the model
- *Clear Model Output* ⇒ Clears the model output
- *Cut* ⇒ Cuts the model for pasting elsewhere
- *Copy* ⇒ Copies the model for pasting elsewhere
- *Delete* ⇒ Deletes the model from the project/integrated model
- *Rename* ⇒ Rename the integrated model
- *Import . . .* ⇒ Opens selection window for a variety of available importers (if present)
- *Export . . .* ⇒ Opens selection window for a variety of available exporters (if present)
- *Expand All . . .* ⇒ Expands all model items
- *Collapse All . . .* ⇒ Collapses all model items
- *Properties* ⇒ Switches to **Properties** window of active integrated model
- *Validate* ⇒ Opens the validation report of the model

As can be seen in Figure 3.19, context menus on the model level can also contain model specific choices. These choices are described in the user manuals for the specific models.

3.3.1.5 Other items

For the other items in the project explorer the following holds:

- Folder items have the same context menu as the project level context menu depicted in Figure 3.16.
- All other project items have context menus similar to the context menu on the region level as shown in Figure 3.18

3.3.2 Main (Central Map window)

The **Main** or **Central Map** window may consist of multiple tabs, ranging from the region editor to table or chart editors or views. A description of the various context menus is given in this section.

3.3.2.1 Region editor

Without any region objects the region editor context menu only consists of choices regarding the underlying map layer, see Figure 3.20:

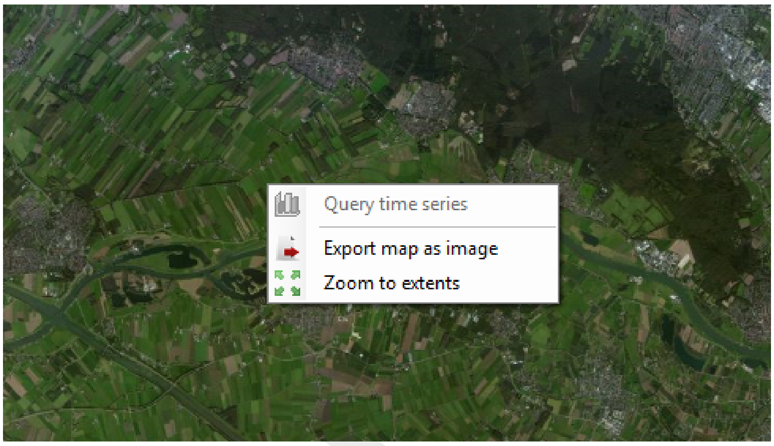


Figure 3.20: *The context menu of the central map window without any region objects.*

- ◇ *Export map as image*
- ◇ *Zoom to extents*

When the region objects are also projected on the **Central Map** window, the context menu is expanded with the more choices, see for example Figure 3.21:

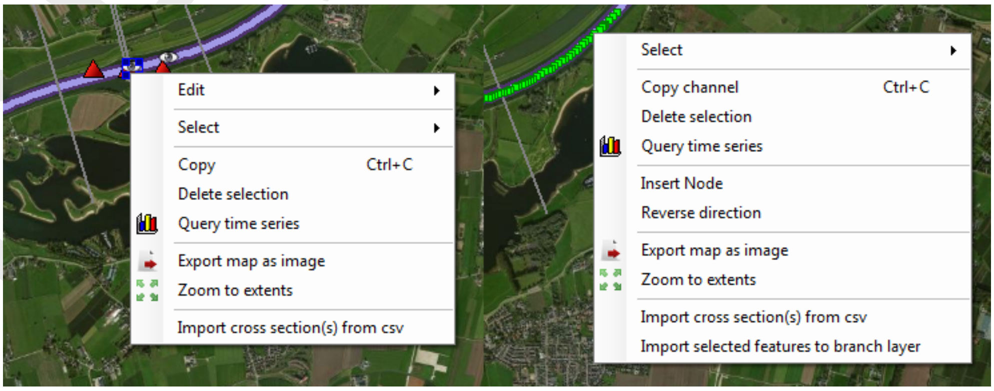


Figure 3.21: *The context menu of the central map window including region objects.*

- ◇ *Edit* ⇒ Edit objects of region
- ◇ *Select* ⇒ Select objects on region
- ◇ *Copy* ⇒ Copy objects on region for pasting elsewhere
- ◇ *Delete selection*
- ◇ *Query time series*
- ◇ *Insert Node*
- ◇ *Remove Node*
- ◇ *Reverse direction*
- ◇ *Import cross section(s) from csv*
- ◇ *Import selected features to branch layer*

3.3.2.2 Table editor

The context menu of the table editors within the **Main** window as shown in Figure 3.22 contains the following choices:

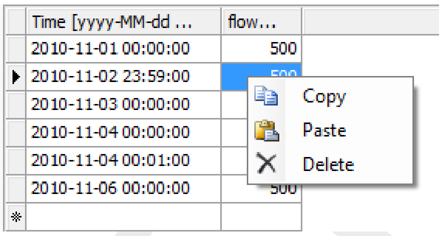


Figure 3.22: *The context menu of the table editor.*

- ◇ *Copy*
- ◇ *Paste*
- ◇ *Delete*

3.3.2.3 Charts

The context menu of the charts within the **Main** window as shown in Figure 3.23 contains the following choice:

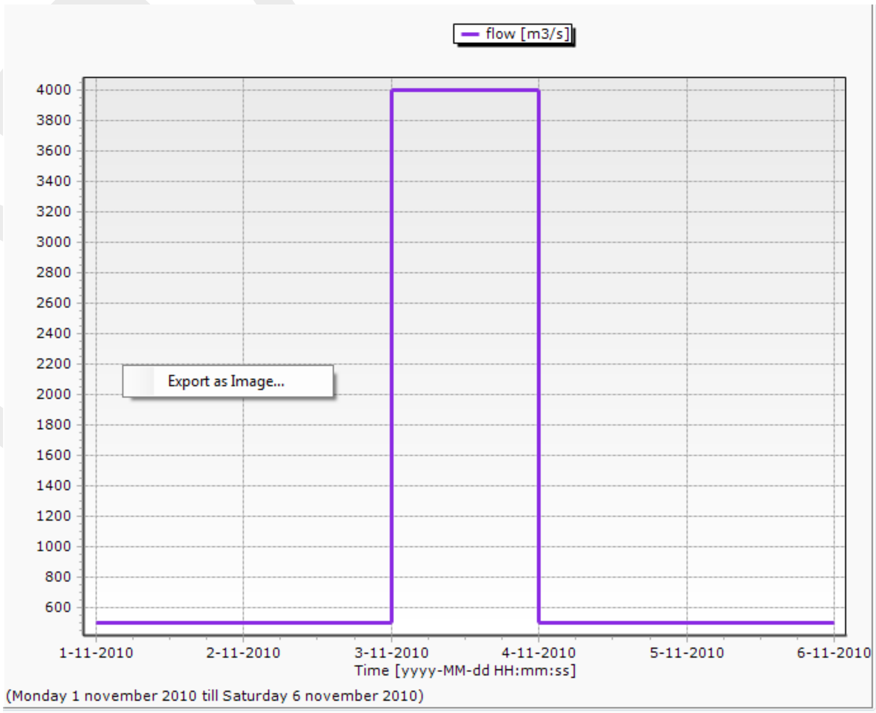


Figure 3.23: *The context menu of the chart view.*

- ◇ *Export as Image . . .*

3.3.3 Map

The context menus for the **Map** window are described in **??**.

3.3.4 Region

Within the **Region** window a variety of context menus is present on different levels within the region tree.

3.3.4.1 Region level

The context menu of the region level within the **Region** window as shown in Figure 3.24 contains the following choice:

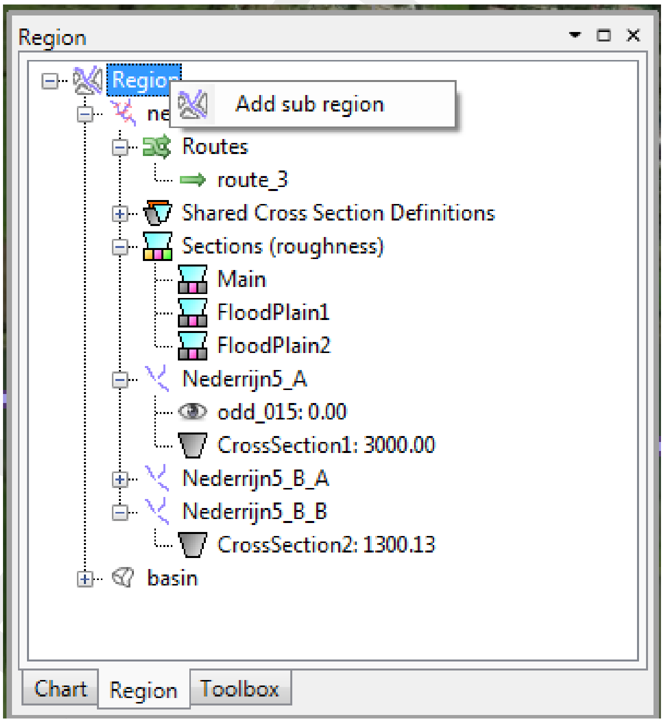


Figure 3.24: *The context menu of the region level within the* ***Region*** *window.*

◇ *Add sub region*

3.3.4.2 Hydro object level

The context menu of the hydro object level within the **Region** window as shown in Figure 3.25 contains the following choices:

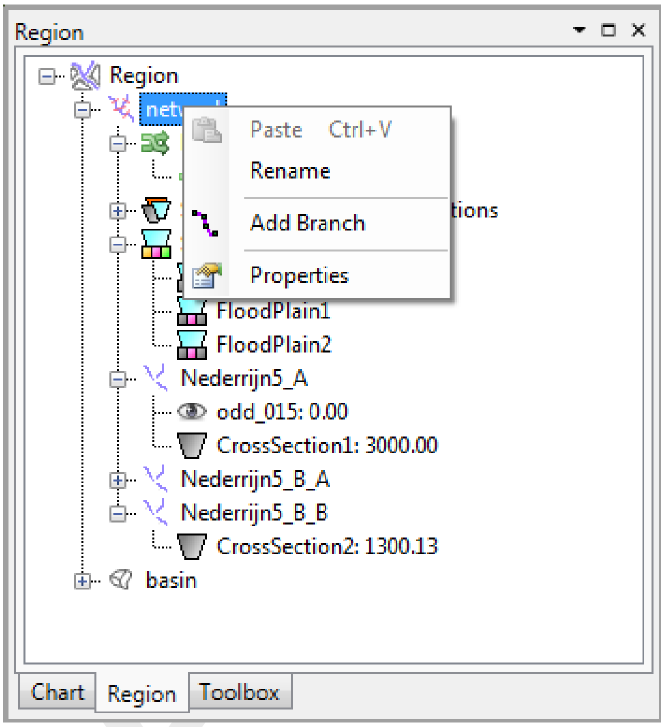


Figure 3.25: *The context menu of the hydro object level within the **Region** window.*

- ◇ *Paste* ⇒ Paste items onto hydro object
- ◇ *Rename*
- ◇ *Add Branch*
- ◇ *Properties*

3.3.4.3 Routes, cross section definitions, roughness sections, branches, catchments

The context menu of the various sub levels of the hydro objects within the **Region** window as shown in Figure 3.26 contains the following choices:

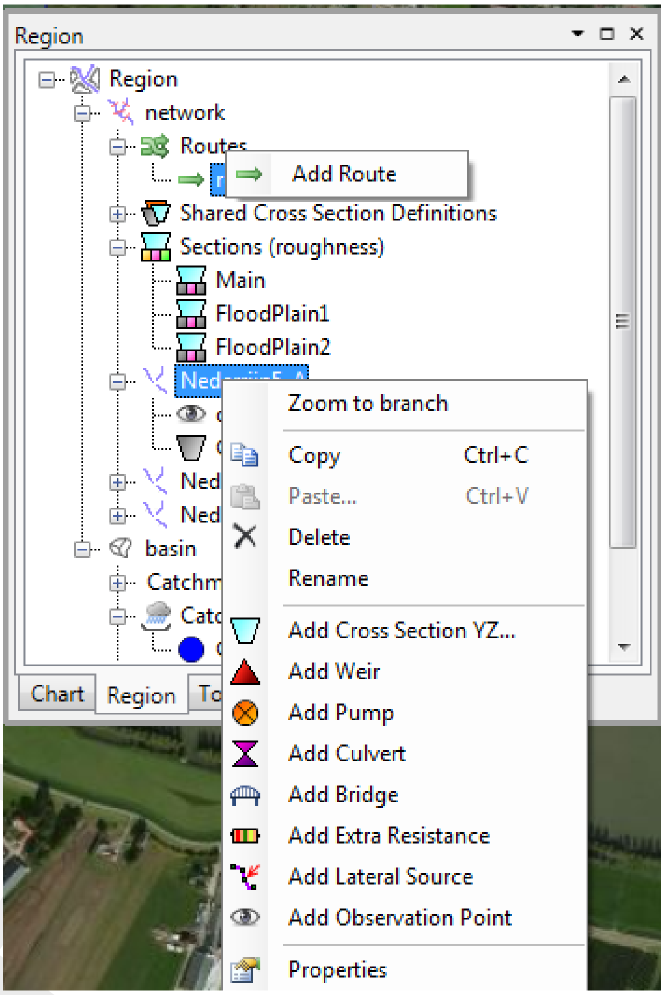


Figure 3.26: *The context menu of the sub levels of the hydro objects within the* ***Region*** *window.*

- ◇ *Zoom to branch*
- ◇ *Copy*
- ◇ *Paste . . .*
- ◇ *Delete*
- ◇ *Rename*
- ◇ *Add . . .* ⇒ Adds items onto hydro object
- ◇ *Properties*

3.3.4.4 Editable items

The context menu of the editable items of the hydro objects within the **Region** window as shown in Figure 3.27 contains the following choices:

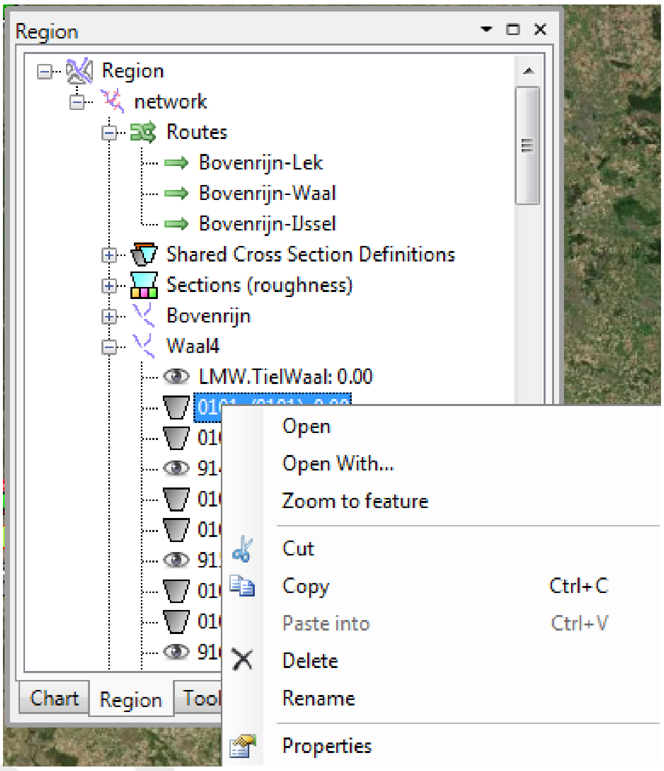


Figure 3.27: *The context menu of the editable items of the hydro objects within the* ***Region*** *window.*

- ◇ *Open*
- ◇ *Open With . . .*
- ◇ *Zoom to feature*
- ◇ *Cut*
- ◇ *Copy*
- ◇ *Paste into*
- ◇ *Delete*
- ◇ *Rename*
- ◇ *Properties*

3.3.5 Toolbox

The context menus for the **Toolbox** window is described in **??**.

3.3.6 Messages

The context menu for the **Messages** window as shown in Figure 3.28 contains the following choices:

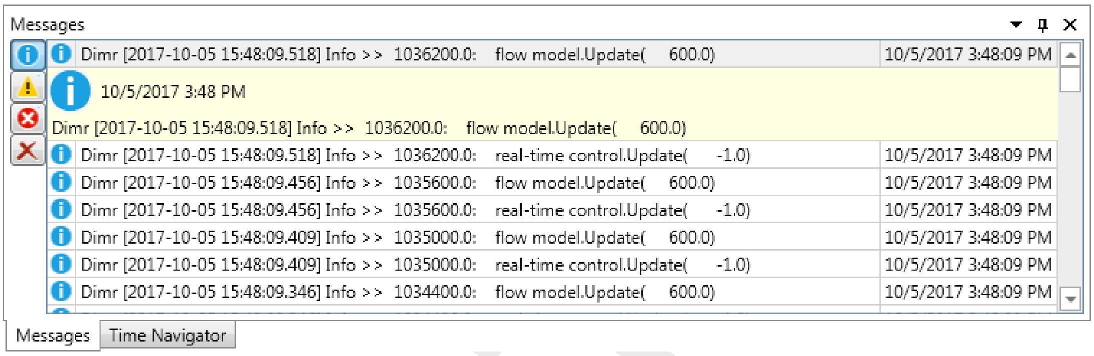


Figure 3.28: *The context menu for the* ***Messages*** *window.*

- ◇ *Copy*
- ◇ *Show details* ⇒ Open separate window with detailed message information
- ◇ *Clear all*

3.4 Ribbons and toolbars

The user can access the toolbars arranged in *ribbons*. Model plug-ins can have their own model specific *ribbon*. The *ribbon* may be auto collapsed by activating the *Collapse the Ribbon* button when right-mouse-clicking on the *ribbon*.

3.4.1 Ribbons (shortcut keys)

The User Interface makes use of ribbons, just like Microsoft Office. You can use these ribbons for most of the operations. With the ribbons comes shortcut key functionality, providing shortcuts to perform operations. If you press `Alt`, you will see the letters and numbers to access the ribbons and the ribbon contents (i.e. operations). For example, `Alt + H` will lead you to the *Home*-ribbon (Figure 3.29).

Note: Implementation of the shortcut key functionality is still work in progress.

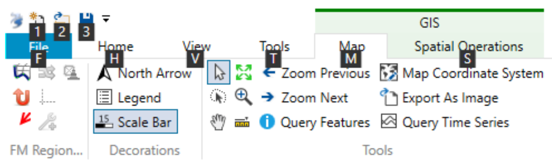


Figure 3.29: *Perform operations using the shortcut keys*

3.4.2 File

The left-most *ribbon* is the *File* ribbon. It has menu-items comparable to most Microsoft applications. Furthermore, it offers users save and open functionality, as well as the *Info* and *Options* dialogs, as shown in Figure 3.30 and Figure 3.31.

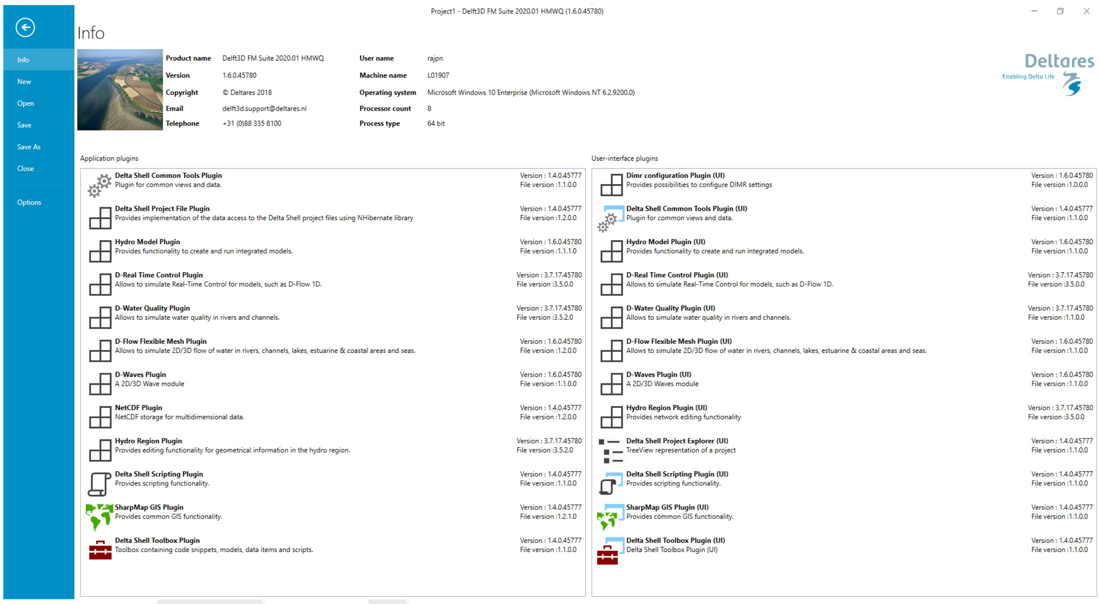


Figure 3.30: *The* File *ribbon.*

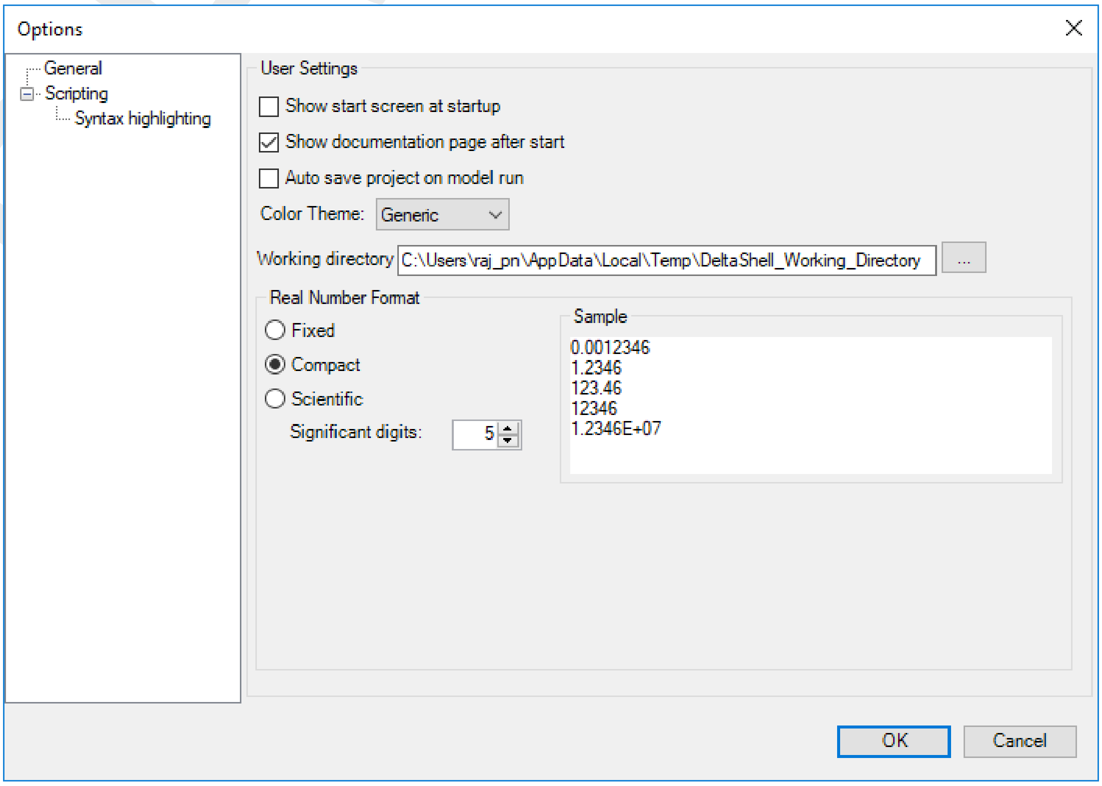


Figure 3.31: *The User Interface options dialog.*

3.4.3 Home

The second *ribbon* is the *Home* ribbon (Figure 3.32). It harbours some general features for clipboard actions, addition of items, running models, finding items within projects or views, and help functionality.

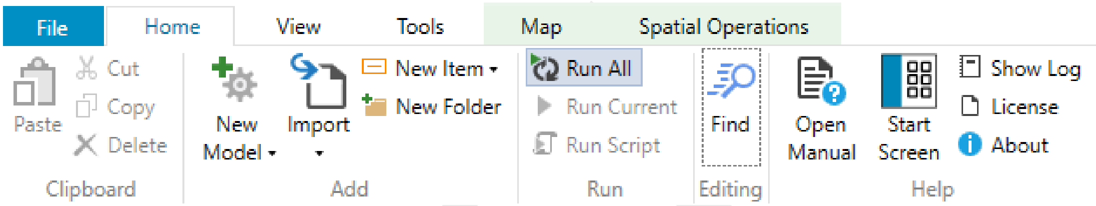


***Figure 3.32:** The* Home *ribbon.*

3.4.4 View

The third *ribbon* is the *View* ribbon (Figure 3.33). Here, the user can show or hide windows.

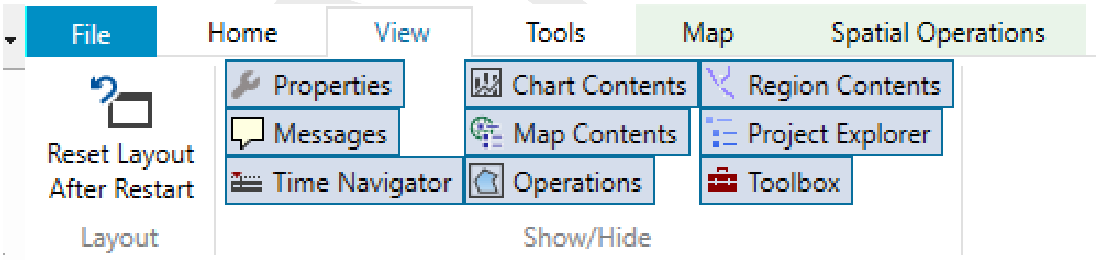


***Figure 3.33:** The* View *ribbon.*

3.4.5 Tools

The fourth *ribbon* is the *Tools* ribbon (Figure 3.34). By default, it contains only the *Open Case Analysis View* tool. Some model plug-ins offer the installation of extra tools that may be installed. These are documented within the user documentation of those model plug-ins.

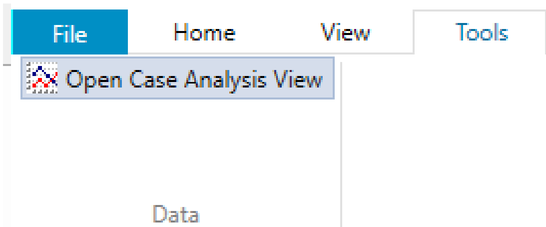


***Figure 3.34:** The* Tools *ribbon contains just the* Data *item.*

3.4.6 Map

The last *ribbon* is the *Map* ribbon (Figure 3.35).

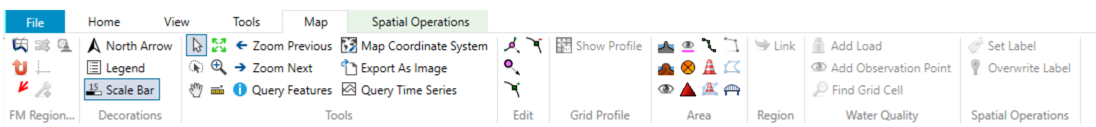


***Figure 3.35:** The* Map *ribbon.*

This will be used heavily, while it harbours all *Geospatial* functions, like:

◇ *Decorations* for the map
  - North arrow
  - Scale bar
  - Legend
  - ...

◇ *Tools* to customize the map view
  - Select a single item
  - Select multiple items by drawing a curve
  - Pan
  - Zoom to Extents
  - Zoom by drawing a rectangle
  - Zoom to Measure distance
  - ...

◇ *Edit* polygons, for example within a network, basin, or waterbody
  - Move geometry point(s)
  - Add geometry point(s)
  - Remove geometry point(s)

◇ Addition of *Area* elements, such as
  - Add 2D Cross-section
  - Add 2D Structure
  - Add 2D Pump
  - ...

Still, all functions of the category can be activated as they will appear in the drop-down panel.

3.4.7 Scripting

When you open the scripting editor in the User Interface, a Scripting *ribbon* category will appear. This ribbon has the following additional options (see also Figure 8.11), which are described in Table 8.1:

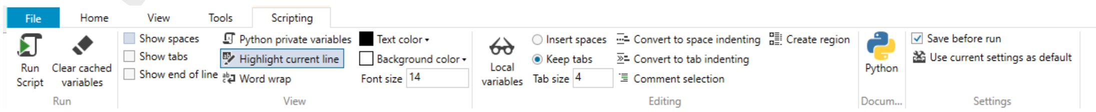


***Figure 3.36:** The scripting* ribbon *within the User Interface.*

***Table 3.1:** Functions and their descriptions within the scripting* ribbon *of the User Interface.*

Function	**Description**
Run script	Executes the selected text. If no text is selected then it will execute the entire script
Clear cached variables	Clears all variables and loaded libraries from memory

***Table 3.1:** Functions and their descriptions within the scripting* ribbon *of the User Interface.*

Function	**Description**
Debugging	Enables/Disables the debug option. When enabled you can add breakpoint to the code (using `F9` or clicking in the margin) and the code will stop at this point before executing the statement (use F10 (step over) or F11 (step into) for a more step by step approach)
Python variables	Show or hide python variables (like _var_) in code completion
Insert spaces/tabs	Determines if spaces or tab characters are added when pressing tab
Tab size	Sets the number of spaces that are considered equal to a tab character
Save before run	Saves the changes to the file before running
Create region	Creates a new region surrounding the selected text
Comment selection	Comments out the selected text
Convert to space indenting	Converts all tab characters in the script to spaces. The number of spaces is determined by Tab size
Convert to tab indenting	Converts all x number of space characters (determined by Tab size) in the script to tabs
Python (documentation)	Opens a link to the python website, showing you the python syntax and standard libraries

3.4.8 Shortcuts

The shortcut keys of the scripting editor within the User Interface are documented in Table 8.2.

***Table 3.2:** Shortcut keys within the scripting editor of the User Interface.*

Shortcut	**Function**
`Ctrl + Enter`	Run selection (or entire script with no selection)
`Ctrl + Shift + Enter`	Run current region (region where the cursor is in)
`Ctrl + X`	Cut selection
`Ctrl + C`	Copy selection
`Ctrl + V`	Paste selection
`Ctrl + S`	Save script
`Ctrl + -`	Collapse all regions
`Ctrl + +`	Expand all regions
`Ctrl + "`	Comment or Uncomment current selection
`Ctrl + W`	Add selection as watch variable
`Ctrl + H`	Highlight current selection in script (press esc to cancel)
`F9`	Add/remove breakpoint (In debug mode only)

Table 3.2: Shortcut keys within the scripting editor of the User Interface.

Shortcut	Function
`F5`	Continue running (In debug mode only — when on break-point)
`Shift + F5`	Stop running (In debug mode only — when on breakpoint)
`F10`	Step over current line and break on next line (In debug mode only - when on breakpoint)
`F11`	Step into current line if possible, otherwise go to next line (In debug mode only — when on breakpoint). This is used to debug functions declared in the same script (that have already runned)

3.4.9 Quick access toolbar

Note: The user can make frequently used functions available by a single mouse-click in the *Quick Access Toolbar*, the top-most part of the application-window. Do this by right-mouse-clicking a ribbon item and selecting *Add to Quick Access Toolbar*.

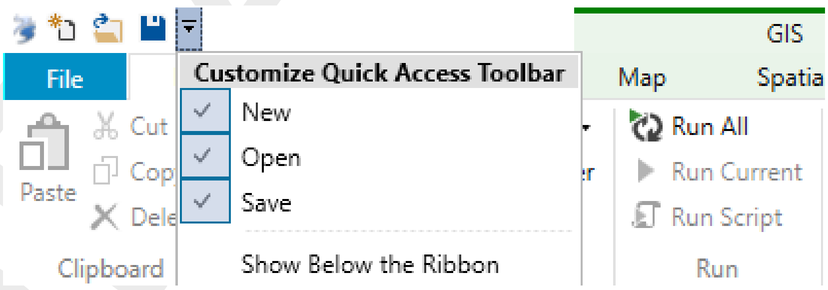


Figure 3.37: *The quick access toolbar.*

4 Case management

Delta Shell currently offers two ways of case management: through the *Project* window or by using the Python scripting functionality (which is described in more detail in chapter 8).

4.1 Case management within the *Project* window

Within a Delta Shell project multiple models can be run and combined. For example multiple runs from a single model, or a model run for a flow model combined with a RTC model or a water quality model. A project may contain different Delta Shell models (see also Figure 4.1), for example one or multiple flow models and a rainfall-runoff model. Model features of one model can be copied or linked to a second model in the project. Complete networks can also be copied or linked. In this way it is easy to create several models with the same basic characteristics, but small differences. In this way the impact of different possible measures on the water system can be analyzed.

Results from different models can be viewed next to each other by docking maps and graphs. Results of different models can also be combined in one single map or graph.

To enhance transparency the **Project** can be sorted by adding folders and moving networks, models and data. It is advised to group several models using one network in a folder to separate them from other models. Figure 4.1 gives an example of a case management in Delta Shell. In this project an original model was used for a sensitivity study and a study for different measures (in this case weirs with or without RTC). The original network was used in the models for the sensitivity study and only the roughness was adjusted. Then, for the measures a new network ("measure1:weirs") was created which was used to model the situation with and without RTC.

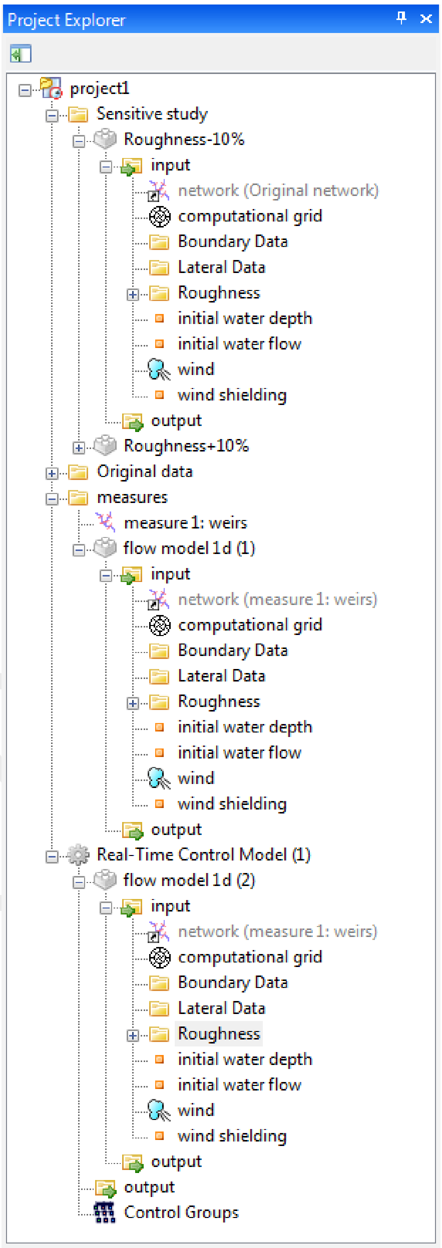


Figure 4.1: *Example of case management*

4.2 Case management using Python scripting

The Python scripting toolbox as provided within the Delta Shell framework can be used for a variety of purposes. One of these is case management. It may be used for sensitivity analyses, batch runs, urban development studies etc.

An example of using Python scripting for case management is presented below. This example script takes a base case existing in the Delta Shell project, then varies the boundary condition on a specific location, runs the model using the new boundary condition, and finally saves the results in a new Delta Shell project.

```
flow = CurrentProject.RootFolder["Base case"]
# Case 1: Batch run
# Changing constant boundaries and save them to completely new projects
constantBoundaries = [1.5, 1.75, 2.0]
```

```
for x in constantBoundaries:
        EditConstantBoundaryCondition(flow, "benedenrand_ZwarteWater", x)
        Application.RunActivity(flow)
        timeseries = GetWaterLevelResultsAtLateralSource(flow, "Zwolle")
        Gui.CommandHandler.OpenView(timeseries)
        projectName = ComposeProjectName("SW_max", x)
        Application.SaveProjectAs(projectName)
```

SOMETHING ABOUT CASE MANAGEMENT AND PYTHON SCRIPTING

5 Working with the map

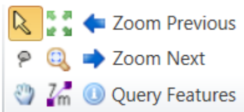
5.1 Introduction

By default all geo-referenced data will be presented in one and the same (tab in the) main window: the Central Map.

It is also possible to use an extra map. The user can add a map to the **Project** by a right-mouse-click on <project1/Add/New item> or a double-click in the **Toolbox** window on *Items/-General/Map*. This map can be customized by the user by dragging individual model features (such as the computational grid, initial conditions, output results etc.) onto the map.

With the left-most part of the *Map* ribbon 

, the user can show or hide the *Scale bar*, *Legend* and *North Arrow*.

The user can zoom and pan by activating one of the buttons 

.

By selecting  distances between two points on the map are shown.

5.2 Map layers

Each layer can be made (in)visible by (un)checking it in the **Map** window. This can be done for the network as a whole for example, but also for the layers within the network, such as *Nodes* or *Weirs*.

The order in which layers are presented in the map is always top-down. The user can adjust this by a right-mouse-click on a layer in **Map** window and selecting for example *Bring to front*.

With the four icons in the top left of the window  new <.shp>- or <.wms>-layers can be added, removed or exported.

By right-mouse-click on a layer in the **Map** window, the following functions are available within the context menu:

- *Open Attribute Table (of GIS-shape)*
- *Cut, Copy, Delete, Rename*
- *Remove Theme*
- *Import legend*
- *Export legend*
- *Order*
  - *Bring to front*
  - *Send to back*
  - *Bring forward*
  - *Send backward*
- *Zoom to Extend*
- *Synchronize zoom level with map*

◇ *Show labels*
◇ *Show in Legend*
◇ *Hide all Layers but this one*
◇ *Properties ...*

By selecting *Properties ...* a window pops up in which symbols, sizes, colors etc. can be adjusted to preference, see also Figure 5.1.

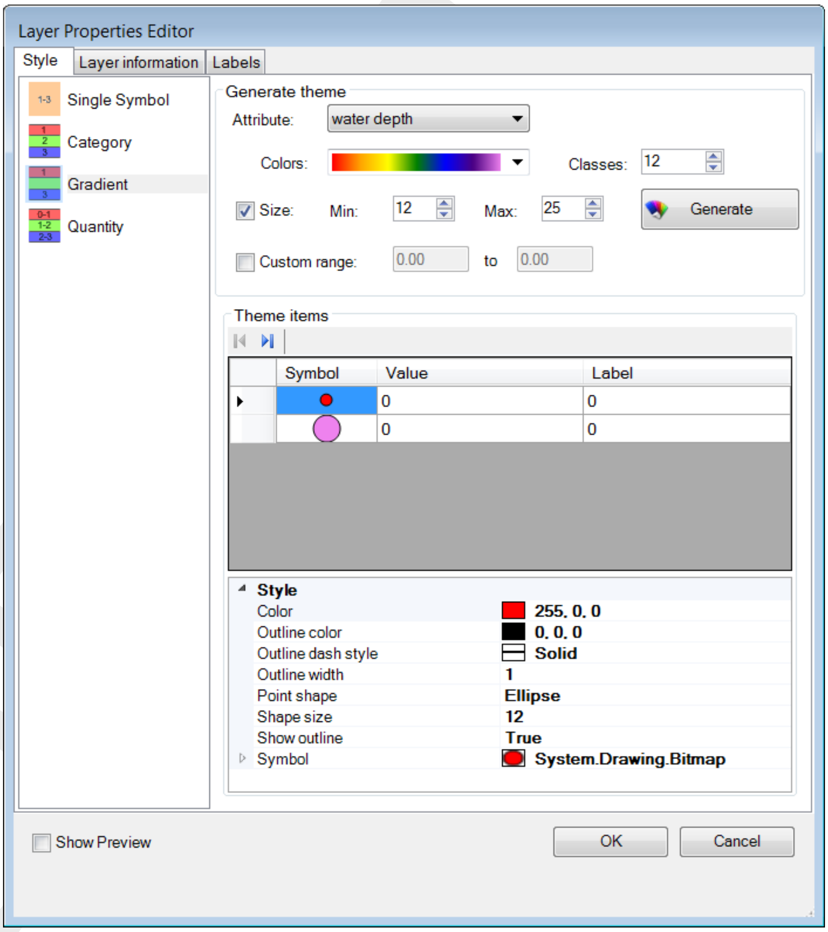


Figure 5.1: *The layer* ***Properties*** *editor.*

By selecting *Open Attribute Table*, a table-view will emerge below the Central Map. With the help of filters the user can search for model objects and edit attribute data. Below the table are buttons to add or delete objects. The table supports the following context menu actions:

◇ *Copy*
◇ *Paste*
◇ *Delete*
◇ *Add attribute*
◇ *Zoom to item*
◇ *Open View ...*

Selecting *Open View ...* will present the editor in a different tab in the main window. Double-clicking on an item in the **Project** window which is not geo-referenced, will also open the editor.

5.3 Background layer

For modelling and presentation a map in the background is very useful, for example a <.shp>-file or <.wms>-layer.

As an example, to set OpenStreetMap as background, the user must:

- ◇ right-mouse-click on <project1> in the **Project** window, and select “Add” and “New Item . . . ”
- ◇ select *General* and *Map(world)*
- ◇ and finally right-mouse-click on the map in the **Project** window, and select *Use as default background layer*

Note: Mark that automatically the coordination system for the map is set on <WGS 84 / Pseudo-Mercator>. More on coordinate systems in the next paragraph (section 5.4).

5.4 Coordinate systems and transformations

The map and all spatial objects in your model, like a network, basin or waterbody, may have a coordinate system associated with it. When presenting the objects in the map, Delta Shell performs the appropriate coordinate transformation (from the coordinate system of the layer to the coordinate system of the map).

The user can specify or adjust the coordinate system for the map by selecting Map Coordinate System in the *Map* ribbon or by a right-mouse-click on *map* in the **Map** window and selecting *Change Map Coordinate System . . .* .

The user can also specify or adjust the coordinate system for model objects by selecting the object in the **Project** window and select *CoordinateSystem* in the **Properties** window, as in Figure 5.2.

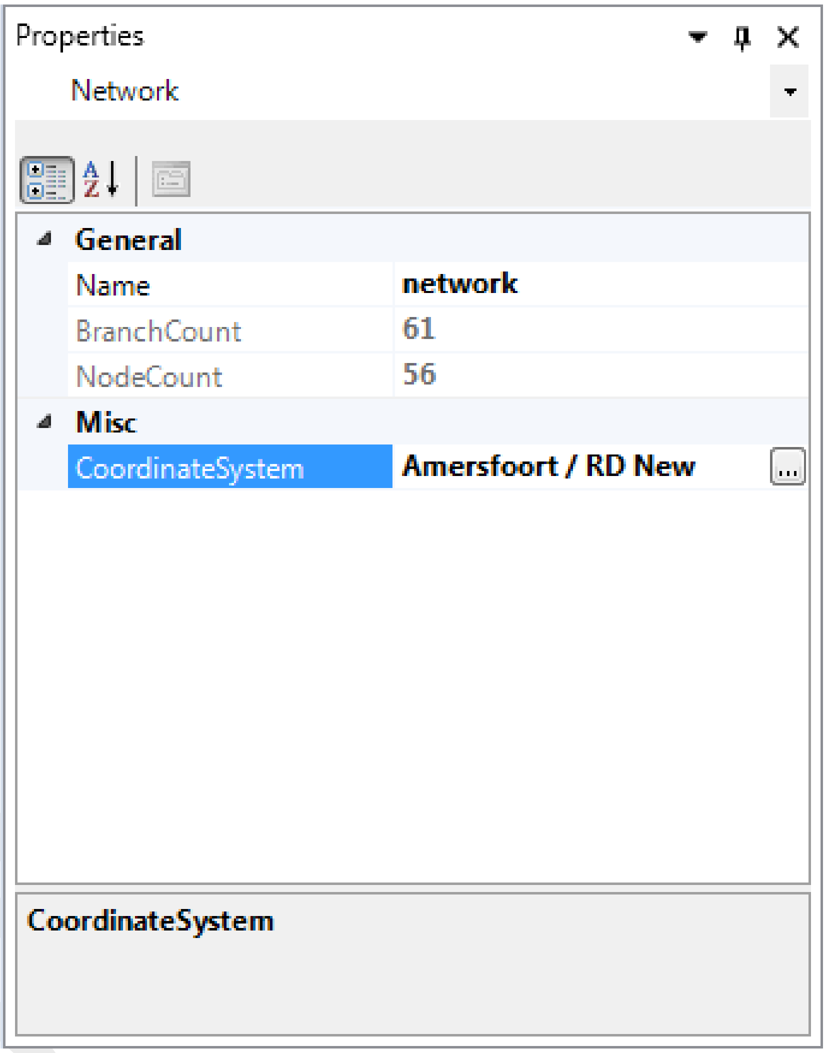


Figure 5.2: *Specify the coordinate system for a network.*

For both ways, the following window will pop up:

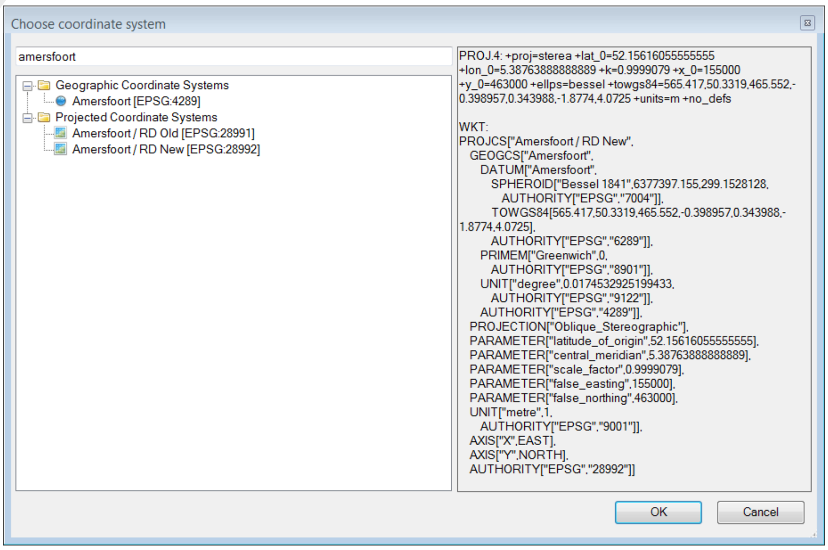


Figure 5.3: *Choose coordinate system.*

Here, the user can select from a very extensive list of coordinate systems. Delta Shell works with open source software provided by the Open Source Geospatial Foundation (http://www.gdal.org/).

By typing in the selection-box the list will be filtered and selection made easier. In the box on the right hand side of the window characteristics of the coordinate system are presented.

6 Spatial editor

6.1 Introduction

The spatial editor is a generic feature of the User Interface for editing spatial data (e.g. bed level, roughness, viscosity, initial conditions, sediment availability). The spatial editor supports both point clouds and coverages (e.g. data on a grid or network). Therefore, you can use the spatial editor both to edit spatial data in general and to prepare model input. This Chapter describes the general features of the spatial editor (section 6.2), (spatial) quantity selection (section 6.3), geometry operations (section 6.4), spatial operations (section 6.5) and the operation stack (section 6.6). The examples given below are typically focusing on use of the spatial editor in the D-Flow FM User Manual.

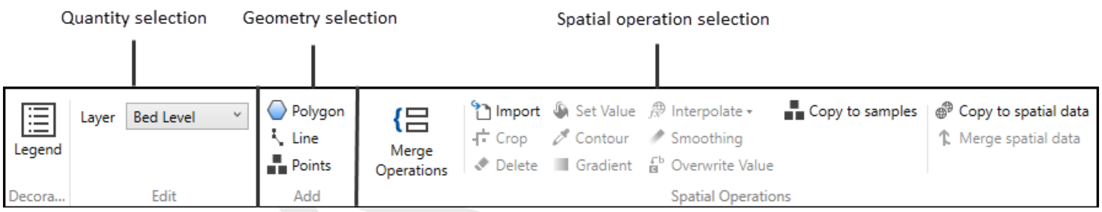


Figure 6.1: *Overview of spatial editor functionality in Map ribbon*

6.2 General

6.2.1 Overview of spatial editor

The spatial editor functionality can be accessed from the “Spatial Operations” ribbon (Figure 6.1). Typically, you first select which data set or quantity (either a point cloud or a coverage) to work on (e.g. bed level, roughness, viscosity, initial conditions, sediment availability), then a geometry (e.g. point, line, polygon) and finally which spatial operation to perform (e.g. crop, delete, set value, contour, interpolate, smoothing). Typical workflows are as follows:

Working on a point cloud dataset:

1. Import the dataset as point cloud (section 6.2.2)
2. Activate/select the dataset (quantity) in the spatial editor (section 6.3)
3. Select a geometry to perform an operation on (section 6.4)
4. Select the spatial operation for this geometry (section 6.5)
5. Repeat steps 3 and 4 until you are satisfied with the data
6. Export the dataset (section 6.6.7)

Working on a coverage (e.g. model input):

1. Activate/select the spatial quantity to work on in the spatial editor (section 6.3)
2. Optional: import a dataset as point cloud (section 6.5.1)
3. Select a geometry to perform an operation on (section 6.4)
4. Select the spatial operation for this geometry (section 6.5)
5. Repeat steps 3 and 4 until you are satisfied with the data
6. Interpolate the point cloud to the grid or network (section 6.5.10)
7. Optional: export the dataset (section 6.6.7)

Upon performing a spatial operation, the ‘Operations’ panel will open (see Figure 6.53) with the operations stack. This stack keeps track of the workflow of spatial operations that you performed. This helps you to make transparent how you arrived at your ‘final’ dataset without having to save all the intermediate datasets (steps) separately. Moreover, the stack is reproducible and easily editable without having to start all over again. When working on a coverage (e.g. the second workflow), point clouds can be used to construct the coverage. In this case the coverage (for example ‘Bed level’) is the ‘trunk’ of the workflow and the point clouds (appearing as sets in the stack) are ‘branches’ or subsets of this trunk (see Figure 6.53). By selecting the set or coverage in the ‘Operations’ panel you determine on which dataset you are working. The interpolate operation (section 6.5.10) allows you to bring data from the point cloud (branch) to the coverage (trunk). For more information on the stack you are referred to section section 6.6.

6.2.2 Import/export dataset

To import a (point cloud) dataset use the context menu on “project” in the “Project Tree”, select “import” from the context menu and select the option "Points from XYZ-file" (Figure 6.2). There is yet another method to import the point cloud. Click on "Import" icon under the Home ribbon to obtain a drop-down menu with the list of importers. Select the option "Points from XYZ-file" under Spatial data section (Figure 6.3) to launch the import wizard.

After the import, the point cloud will be added to the project tree. To activate the point cloud in the spatial editor, either double click the dataset in the project tree (Figure 6.4) or select it from the drop down box in the spatial editor ribbon (Figure 6.5).

Note: Exporter still to be implemented

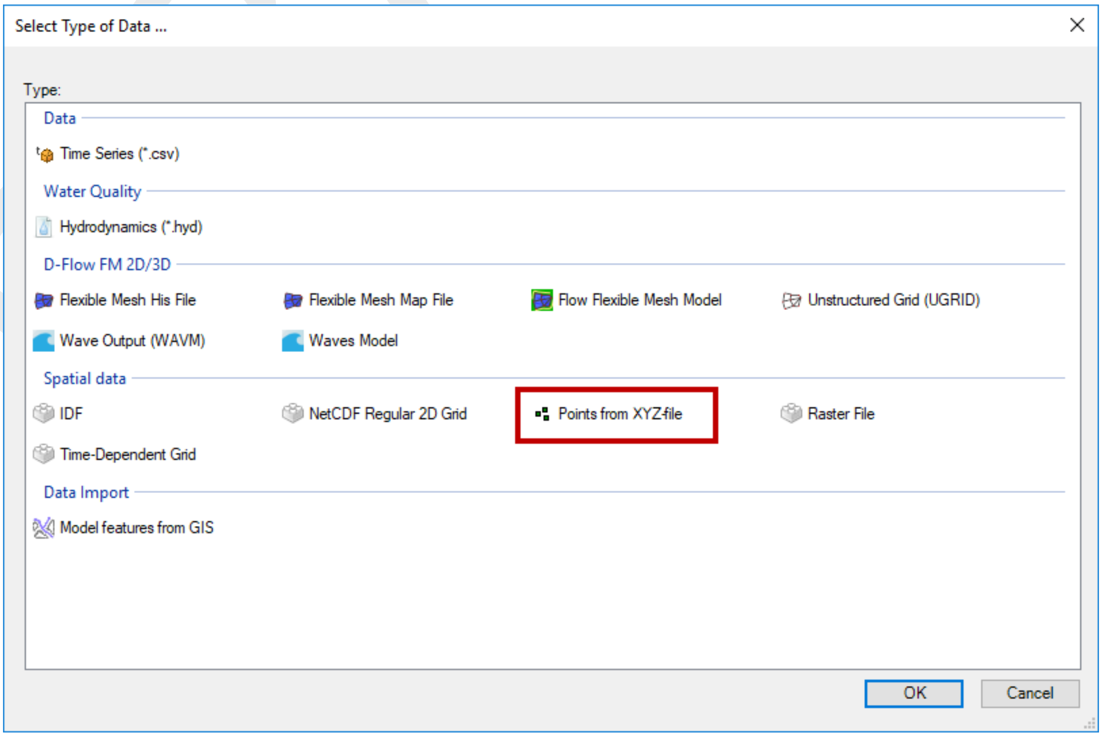


Figure 6.2: *Importing a point cloud into the project using the context menu on “project” in the project tree*

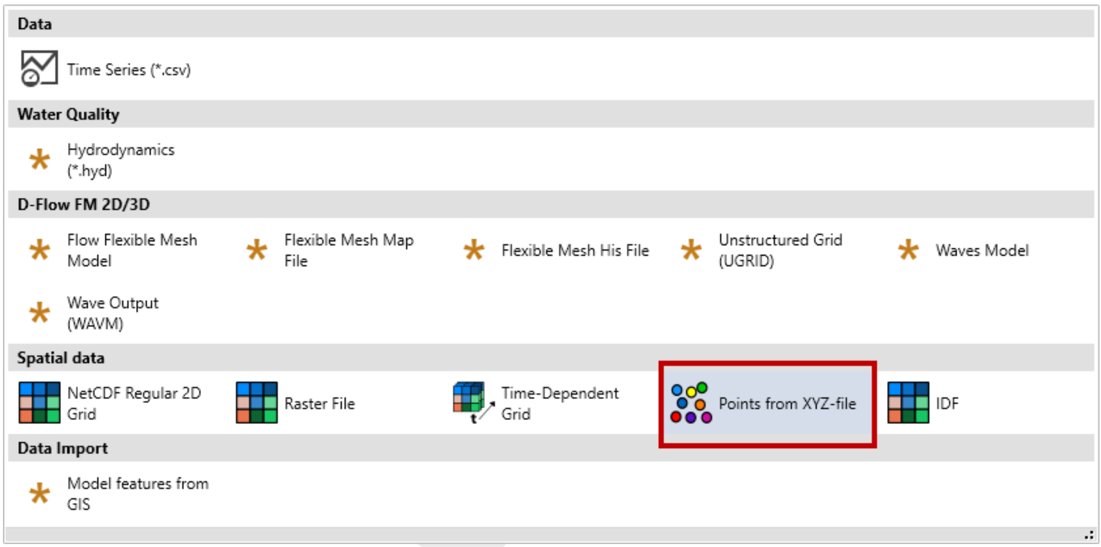


Figure 6.3: *Importing a point cloud into the project using the "Import" drop-down menu in the Spatial Operations ribbon*

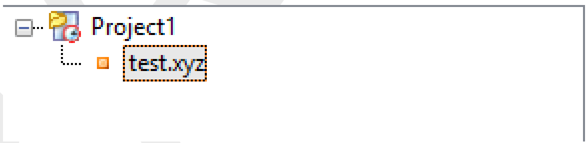


Figure 6.4: *Activate the imported point cloud in the spatial editor by double clicking it in the project tree*

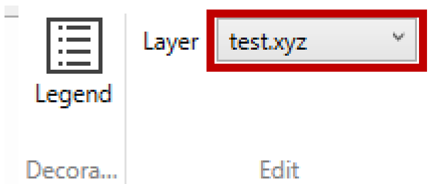


Figure 6.5: *Activate the imported point cloud in the spatial editor by selecting it from the dropdown box in the Map ribbon*

6.2.3 Activate (spatial) model quantity

Similar to activating an imported dataset in the spatial editor, you can also activate a (spatial) model quantity (e.g. bed level, initial conditions, roughness, viscosity) in the spatial editor by double clicking the quantity in the project tree or selecting it from the dropdown box in the "Spatial Operations" ribbon.

6.2.4 Colorscale

In the spatial editor the colorscale for a spatial quantity can be made visible in the map window by clicking on the ”Legend" button in the “Spatial Operations” ribbon (Figure 6.6). Then, in the Map window a colorscale is activated (Figure 6.7). You can (de-)activate the color scale by clicking on the “Legend” button again. By default the colorscale is ranging from the minimum to the maximum value of the active dataset.

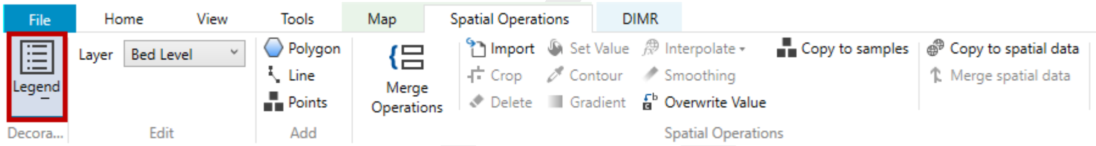


Figure 6.6: *Legend button in Map ribbon*

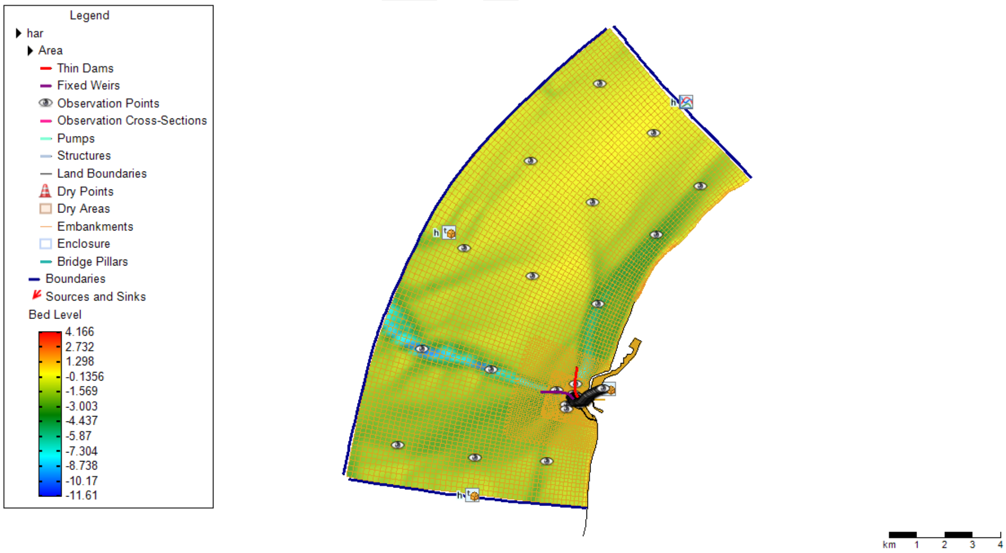


Figure 6.7: *Activate the colorscale by using the Legend in the map ribbon and the colorscale will become visible in the map window.*

You can adjust the colormap and classes of the colorscale using the context menu on the spatial quantity in the “Map tree” and selecting “Legend Properties” (Figure 6.8 left panel). A layer properties editor will open in which you can set the colormap to your own preferences (Figure 6.8 right panel).

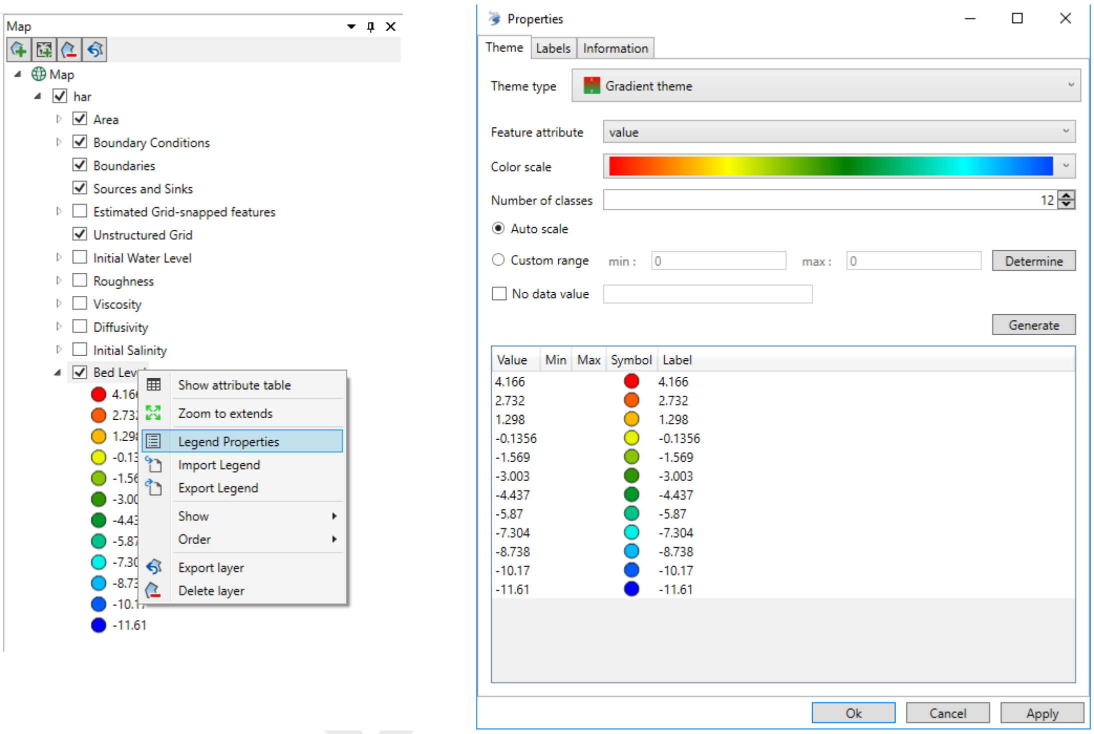


Figure 6.8: *Edit the colorscale properties using the context menu on the active layer in the Map Tree*

6.2.5 Render mode

By default point clouds are rendered as (colored) points and coverages as shades (e.g. ‘FillSmooth’). The render mode can be edited using the properties of the active layer Figure 6.9. The User Interface offers the following render modes:

- Points
- Lines (only for coverages)
- Shades or ‘FillSmooth’ (only for coverages)
- Colored numbers
- Mono colored numbers

An example of a coverage rendered as colored numbers is given in Figure 6.10.

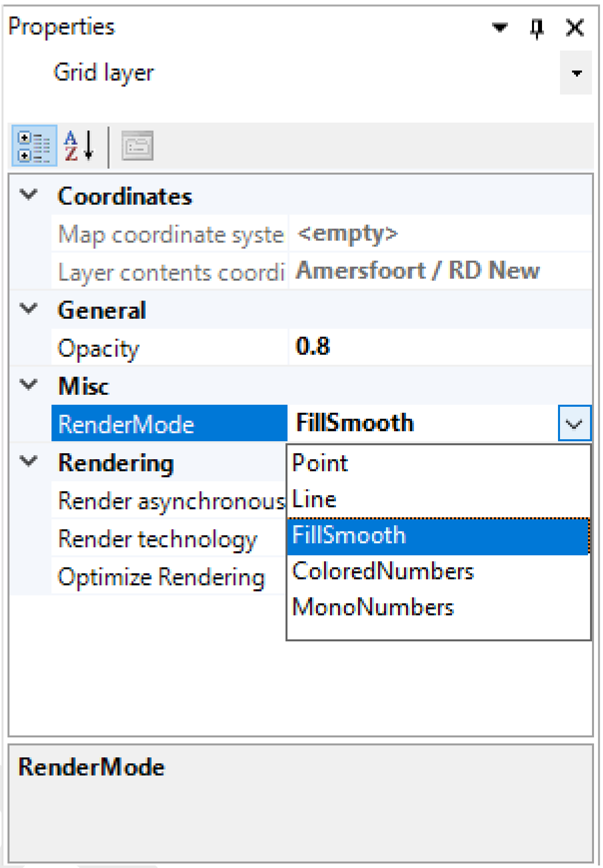


Figure 6.9: *Select the rendermode for the active layer in the property grid.*

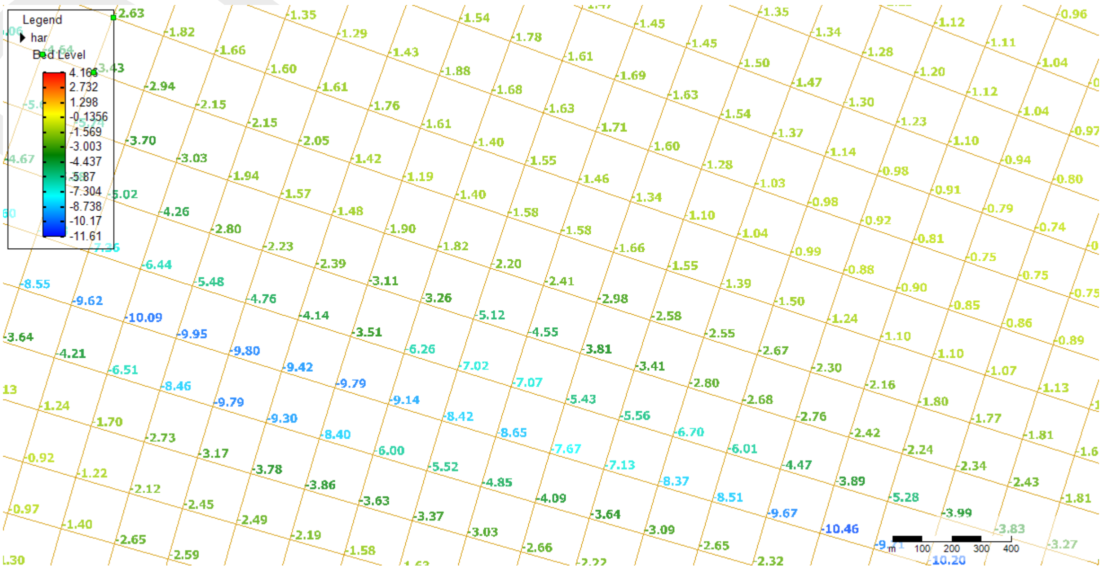


Figure 6.10: *Example of a coverage rendered as colored numbers.*

6.2.6 Context menu

In addition to the selection of spatial operation from the ‘Spatial Operations’ ribbon (see section 6.5), you can also select spatial operations using the context menu (e.g. context menu). After drawing a geometry and clicking the context menu all spatial operation available for the geometry will pop-up (see Figure 6.11). The spatial operation will become active upon selecting it from the context menu.

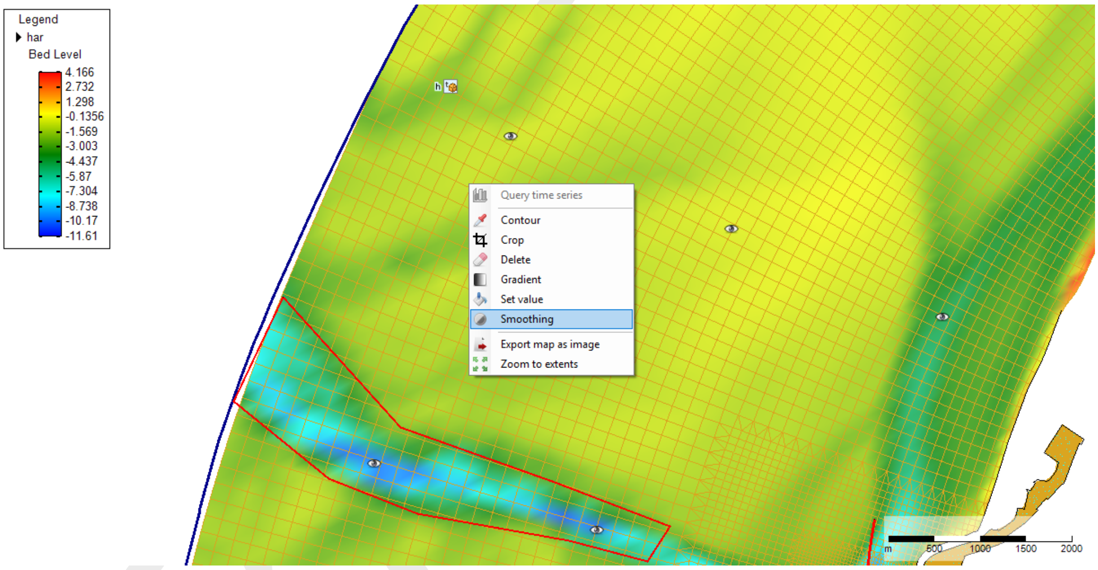


Figure 6.11: Selecting a smoothing operation for a polygon geometry from the context menu (using context menu)

6.3 Quantity selection

A spatial quantity can be activated/selected either by double clicking it in the project tree (Figure 6.12) or by selecting it from the dropdown box in the “Spatial Operations” ribbon (Figure 6.13). Upon selecting the spatial quantity it will be shown as a point cloud (for a dataset) or coverage (for model input) on the central map. Typically, you start from a point cloud (either obtained from import or by generating samples yourself) which will eventually be interpolated to a grid or network (e.g. coverage). The spatial editor will keep track of both the changes made to the point cloud(s) and coverage of the selected spatial quantity. The information will be saved in the User Interface project and available the next time you open the project. **Note:** The operations are not yet saved in a human-readable/editable file

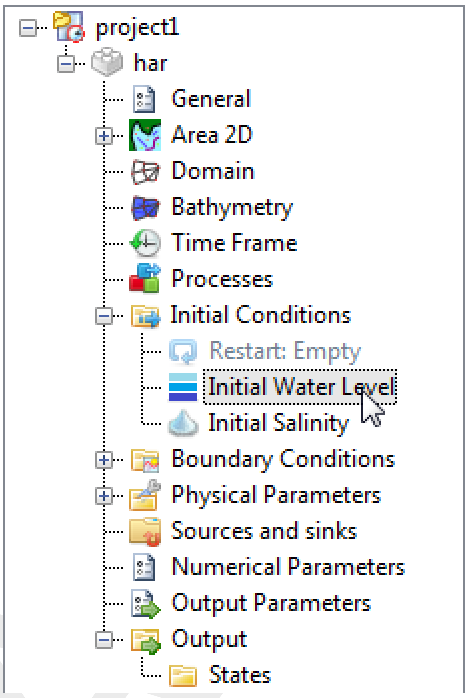


Figure 6.12: *Activating a spatial quantity by double clicking it in the project tree (in this example 'Initial Water Level')*

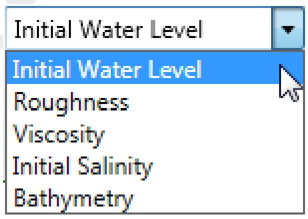


Figure 6.13: *Activating a spatial quantity by selecting it from the dropdown box in the 'Spatial Operations' ribbon*

6.4 Geometry operations

The spatial editor supports three types of geometry operations: (1) polygons, (2) lines and (3) points (see also Figure 6.14). The following sub-sections subsequently describe how these geometries can be selected and edited. If you do not select any of these three geometry operations, the spatial operation automatically applies to all the data.

Note: Please note that the drawn geometries are not yet persistent, implying that the geometries once drawn cannot be edited yet. Upon pressing the "Esc" button while in editing mode all drawn geometries will disappear.

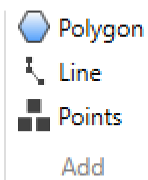


Figure 6.14: *Overview of the available geometry operations in the 'Spatial Operations' ribbon*

6.4.1 Polygons

Upon selecting “Polygon” from the “Map” ribbon you can draw one or multiple polygons (Figure 6.15). Each polygon point is defined by a single click with the LMB. The polygon is closed by double clicking the LMB. After drawing the (first) polygon, the available spatial operations for polygons are enabled in the “Spatial Operations” ribbon. The following spatial operations are available for polygons:

- ◇ Crop (section 6.5.2)
- ◇ Delete (section 6.5.3)
- ◇ Set Value (section 6.5.4)
- ◇ Contour (section 6.5.5)
- ◇ Gradient (section 6.5.9)
- ◇ Smoothing (section 6.5.11)
- ◇ Interpolate - only in case samples and a grid/network are available (section 6.5.10)
- ◇ Copy to samples (section 6.5.6)
- ◇ Copy to spatial data (section 6.5.7) - only for grid coverages

The selected spatial operation applies to all the drawn polygons.

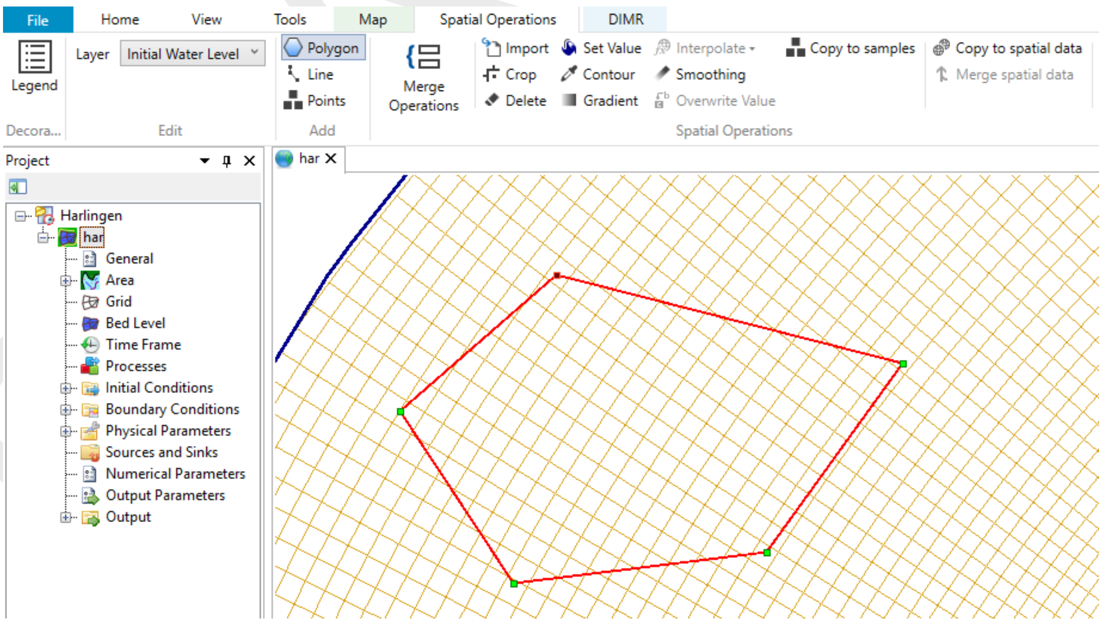


Figure 6.15: *Activating the polygon operation and drawing polygons in the central map.*

6.4.2 Lines

Upon selecting “Line” from the “Map” ribbon you can draw one or multiple lines (Figure 6.16). Each line point is defined by a single click with the LMB. The line is completed by double clicking the LMB. After drawing the (first) line, the available spatial operations for lines are enabled in the “Map” ribbon. The following spatial operations are available for lines:

- ◇ Contour (section 6.5.5)
- ◇ Copy to samples (section 6.5.6)
- ◇ Copy to spatial data (section 6.5.7) - only for grid coverages

The selected spatial operation applies to all the drawn lines.

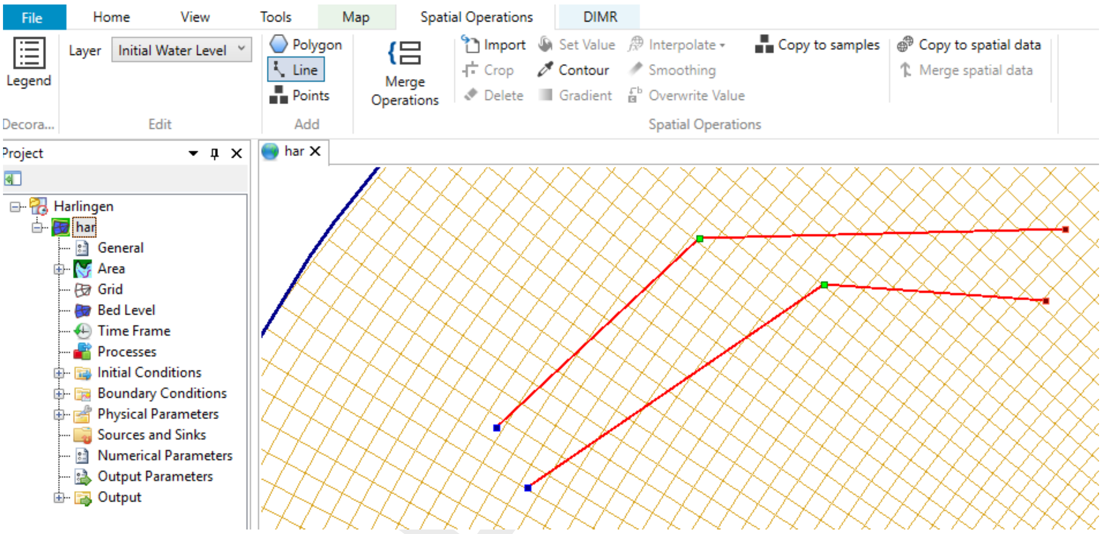


Figure 6.16: *Activating the line operation and drawing lines in the central map.*

6.4.3 Points

Upon selecting "Add points" from the "Map" ribbon you can draw one or multiple points and assign a uniform value to them (Figure 6.17). Each point is defined by a single click with the LMB. The group of points is completed by double clicking the LMB. Upon double clicking a popup appears in which you can assign a value to the points.

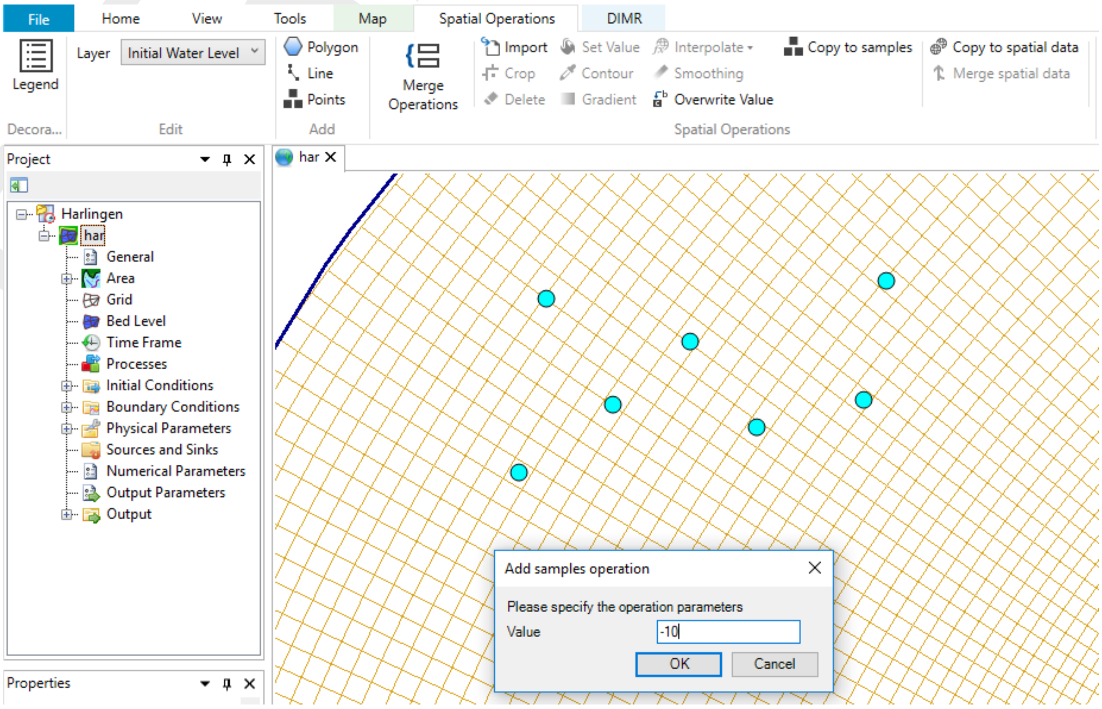


Figure 6.17: *Activating the 'Add points' operation, drawing them in the central map and assigning a value to them.*

6.5 Spatial operations

The spatial editor supports the following spatial operations (see also Figure 6.18):

◇ Import (section 6.5.1) - only for point clouds

- Crop (section 6.5.2)
- Delete (section 6.5.3)
- Set Value (section 6.5.4)
- Contour (section 6.5.5) - only for point clouds
- Gradient (section 6.5.9)
- Interpolate (section 6.5.10) - only for point clouds
- Smoothing (section 6.5.11)
- Change single value (section 6.5.12) - only for grid coverages
- Merge spatial data (section 6.5.8) - only for grid coverages
- Copy to samples (section 6.5.6) - only for grid coverages
- Copy to spatial data (section 6.5.7) - only for grid coverages

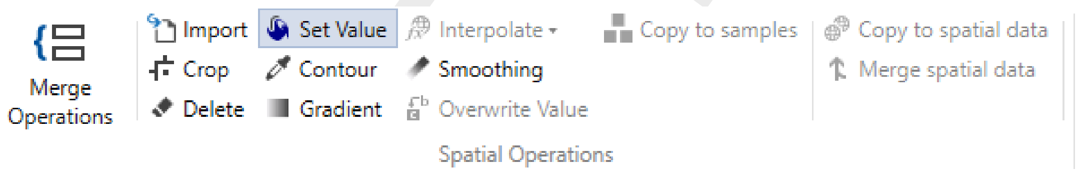


Figure 6.18: *Overview of the available spatial operations in the 'Spatial Operations' ribbon*

The sections below provide a description of each operation.

6.5.1 Import point cloud

With the import operation you can import a point cloud for the selected spatial quantity. For this operation no geometry is required. The import operation is activated from the 'Spatial Operations' ribbon (Figure 6.19). Upon importing a point cloud you are asked whether a coordinate transformation should be applied to the imported dataset (Figure 6.20). By default it will be assumed that the imported data is in the same coordinate system as the model. If not, you can indicate from which to which coordinate system the data should be transformed. After import the point cloud is added to the operations stack (Figure 6.21). The difference between this importer and importing a point cloud on the project level in the project tree (section 6.2.2) is that for this importer the point cloud is directly assigned to the selected spatial quantity (e.g. model input) instead of being treated as a separate dataset.

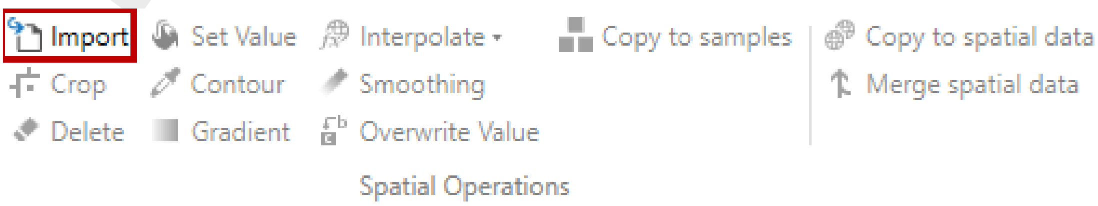


Figure 6.19: *Importing a point cloud using the 'Import' operation from the 'Spatial Operations' ribbon*

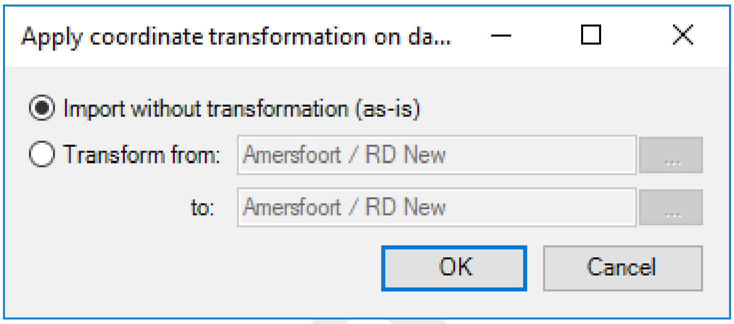


Figure 6.20: *Option to perform a coordinate transformation on the imported point cloud*

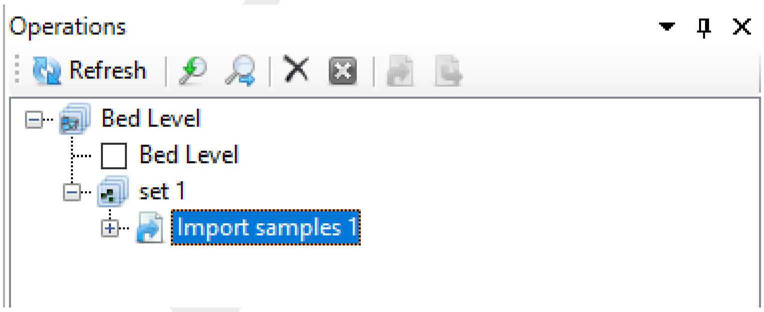


Figure 6.21: *Appearance of import point cloud operation in the operations stack*

6.5.2 Crop

The crop operation crops a point cloud or coverage (depending on which one is active). The crop operation is activated from the 'Spatial Operations' ribbon and only available for polygon geometries. You can control which part of the data should be deleted by using polygons. If you provide a polygon outside the point cloud or coverage, all data will be deleted. For an example see Figure 6.22. After cropping (part of) the point cloud of coverage the operation is added to the operations stack (Figure 6.23).

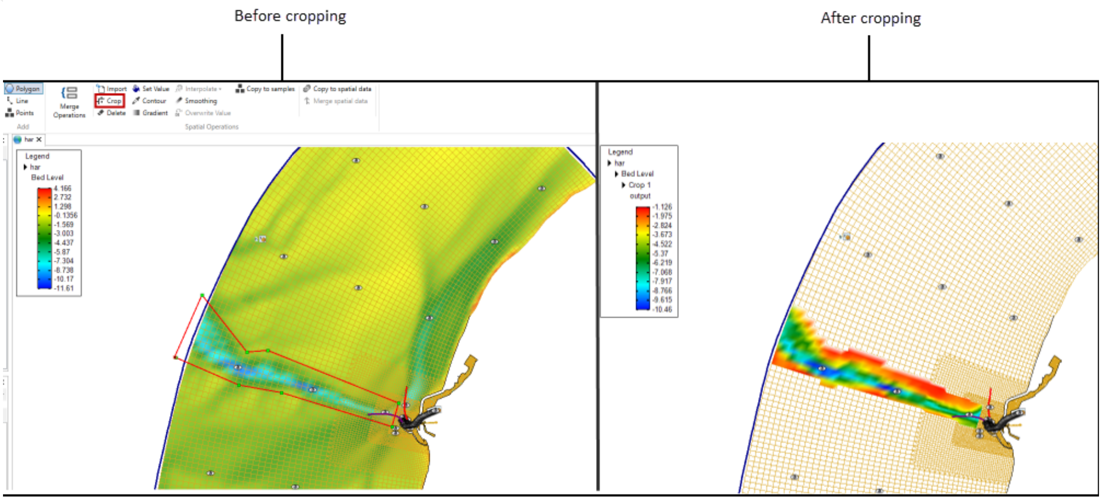


Figure 6.22: *Performing a crop operation on a point cloud with a polygon using 'Crop' from the 'Spatial Operations' ribbon*

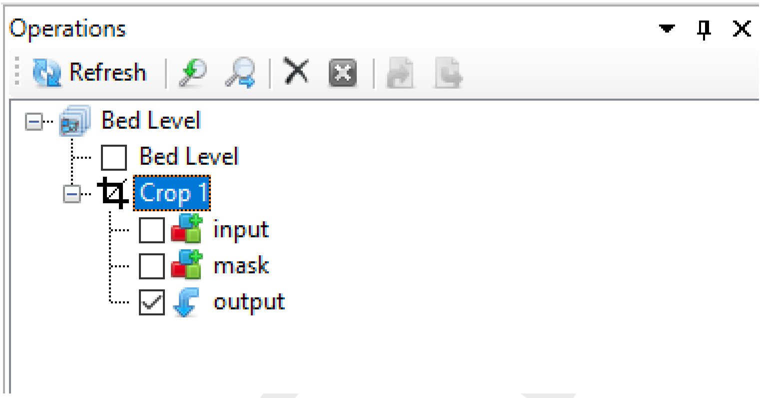


Figure 6.23: *Appearance of crop operation in the operations stack*

6.5.3 Delete

The delete operation deletes (part of) a point cloud or coverage (depending on which one is active). The delete operation is activated from the 'Spatial Operations' ribbon. You can control which part of the data should be deleted by using polygons. If no polygons are provided, the total dataset will be deleted. For an example see Figure 6.24. After erasing (part of) the point cloud or coverage the operation is added to the operations stack (Figure 6.25).

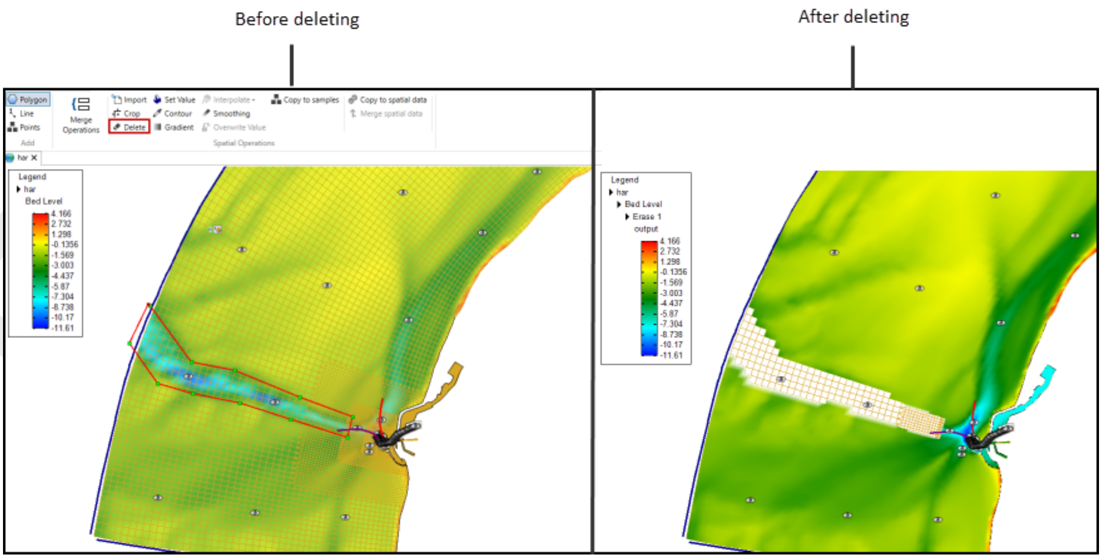


Figure 6.24: *Performing an delete operation on a point cloud with a polygon using 'Delete' from the 'Spatial Operations' ribbon*

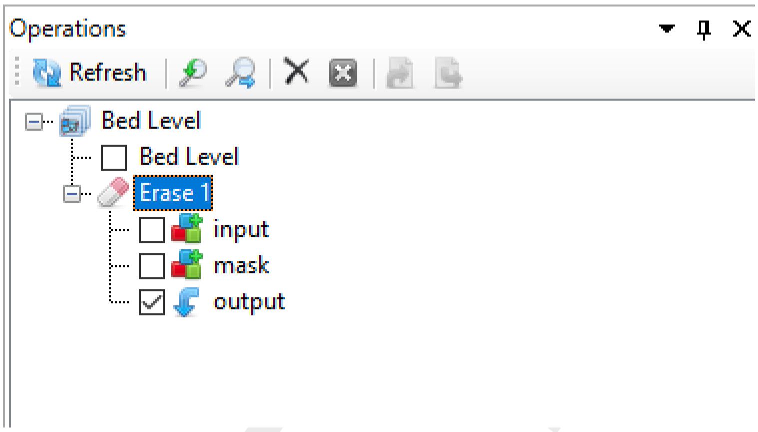


Figure 6.25: *Appearance of delete operation in the operations stack*

6.5.4 Set value

The set value operation assigns a value to a point cloud or coverage (depending on which one is active). The set value operation is activated from the 'Spatial Operations' ribbon and only available for polygon geometries or for the total data set if no polygon is provided. By assigning a value, the user can choose from the following operations:

- **Overwrite** : overwrites all existing points within the polygon (excluding no data values) with the uniform value
- **Overwrite where missing (only for coverages)** : overwrites all missing values within the polygon with the uniform value
- **Add** : Adds the uniform value to all existing points within the polygon (excluding no data values)
- **Subtract** : Subtracts the uniform value from all existing points within the polygon (excluding no data values)
- **Multiply** : Multiplies all existing points within the polygon (excluding no data values) with the uniform value
- **Divide** : Divides all existing points within the polygon (excluding no data values) by the uniform value
- **Maximum** : Sets all existing points within the polygon (excluding no data values) to the maximum of its current value and the uniform value
- **Minimum** : Sets all existing points within the polygon (excluding no data values) to the minimum of its current value and the uniform value

For an example see Figure 6.26. After performing a set value operation to (part of) the point cloud or coverage the operation is added to the operations stack (Figure 6.27).

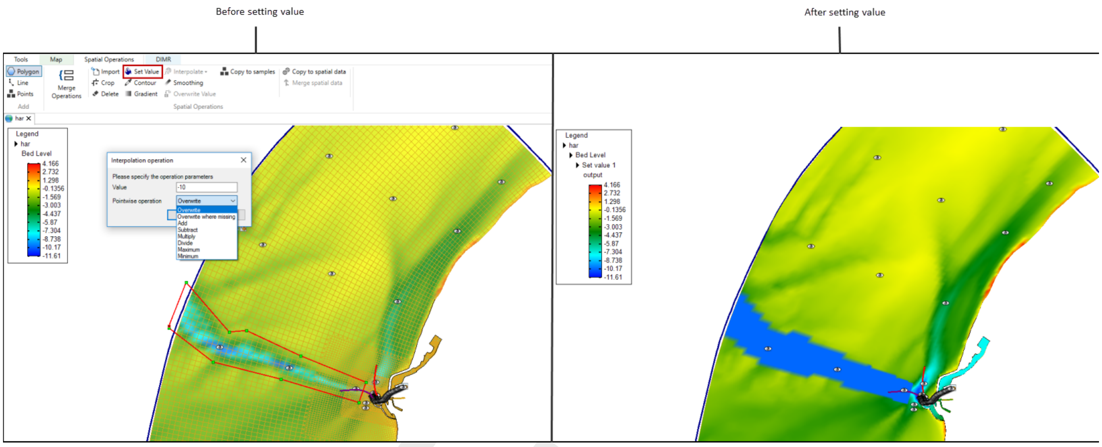


Figure 6.26: *Performing a set value operation (e.g. overwrite) on a point cloud with a polygon using 'Set Value' from the 'Spatial Operations' ribbon*

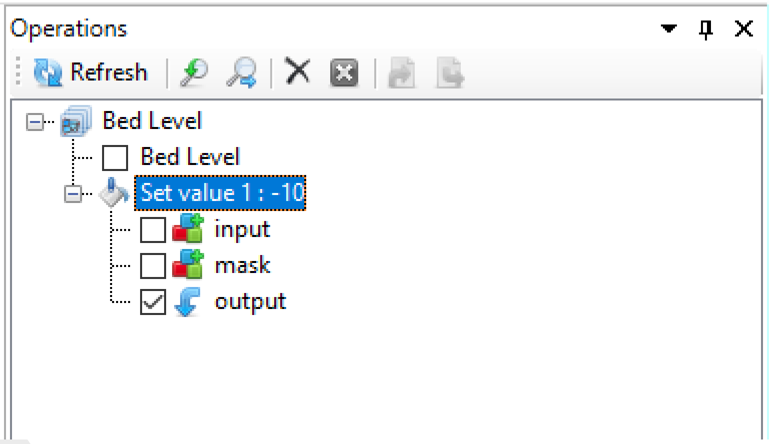


Figure 6.27: *Appearance of set value operation in the operations stack*

6.5.5 Contour

The contour operation creates a point cloud with a uniform value along a line or polygon (depending on which one is active). The contour operation is activated from the 'Spatial Operations' ribbon. After drawing the lines or polygons you have to assign the uniform value (argument) and the sampling interval in m. This spatial operation can be useful to digitalize information from nautical charts for example. In this case you first have to import the nautical chart as a geotiff (Figure 6.28), set the right map coordinate system (Figure 6.29) and then use the contour operation Figure 6.30. Sometimes the samples are created behind the geotiff. Then you can use the context menu on the samples layer in the Map tree to bring the samples to the front (Figure 6.31). After applying the contour operation it is added to the stack (Figure 6.32).

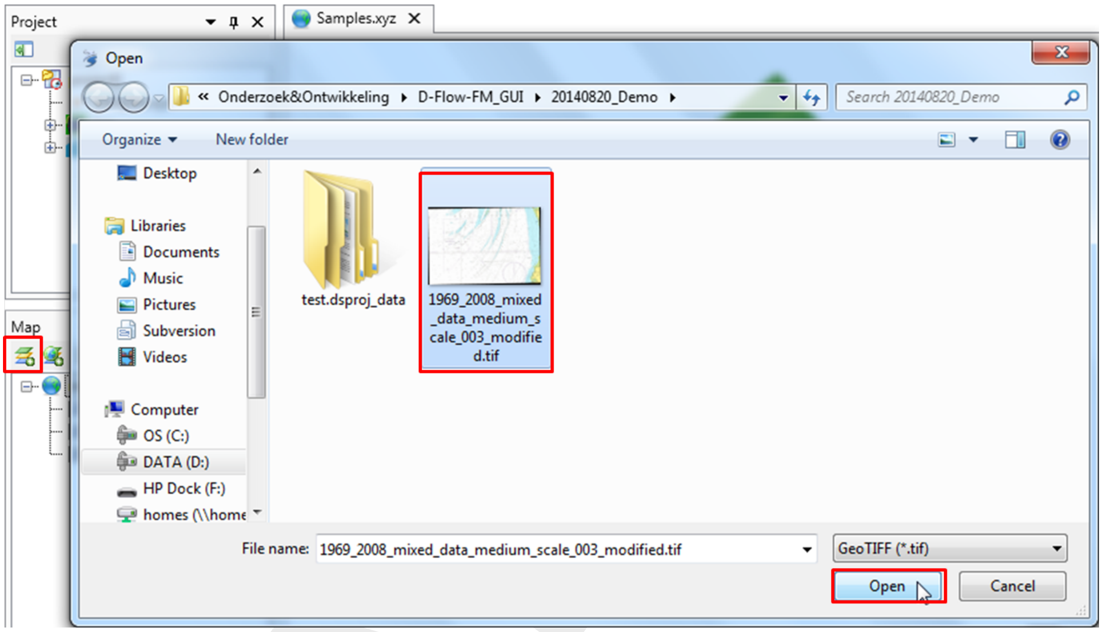


Figure 6.28: *Import a nautical chart as a georeferenced tiff file*

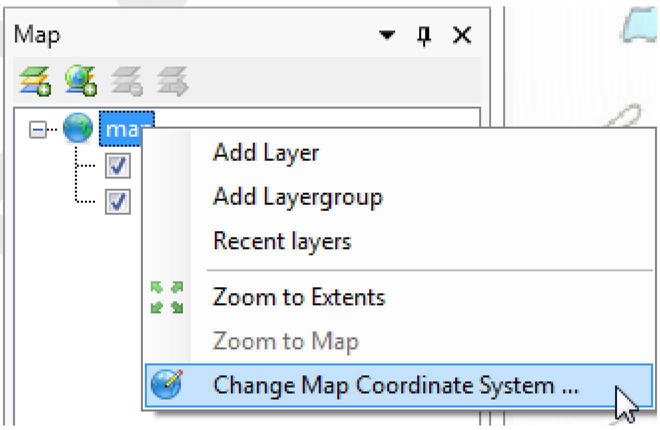


Figure 6.29: *Set the right map coordinate system for the geotiff*

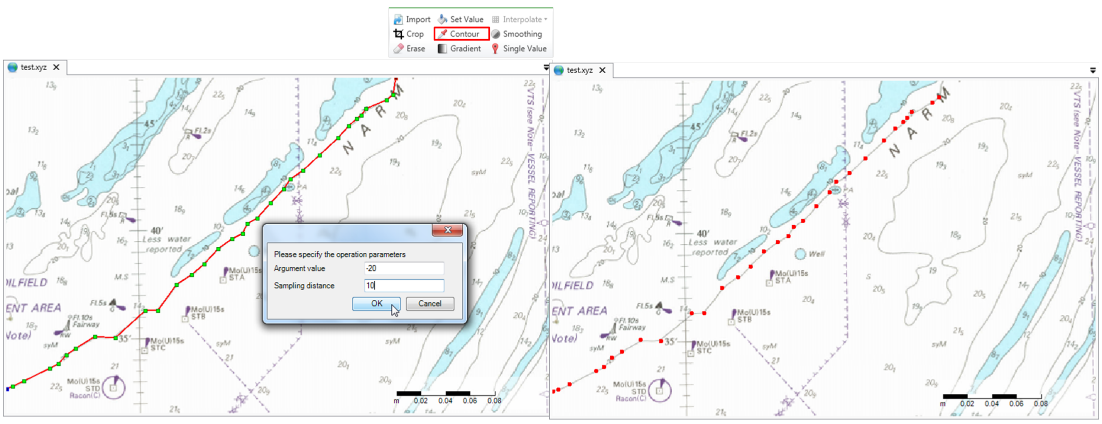


Figure 6.30: *Performing a contour operation on a nautical chart using lines to define the contours and 'Contour' from the 'Spatial Operations' ribbon*

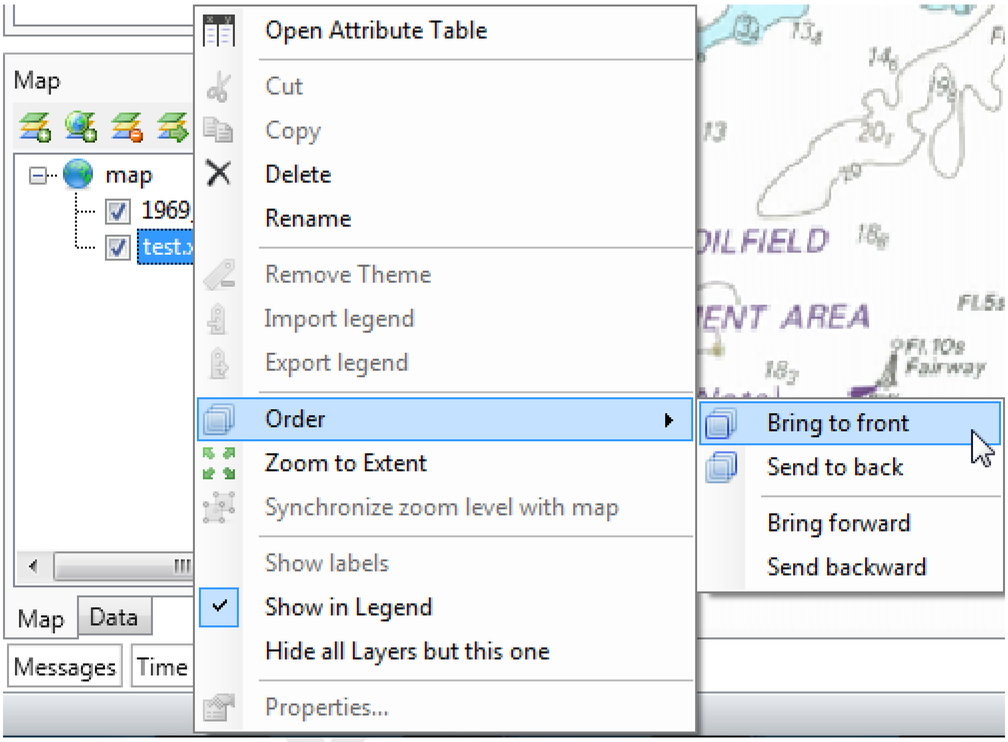


Figure 6.31: *Bring the sample set to the front if it appears behind the nautical chart*

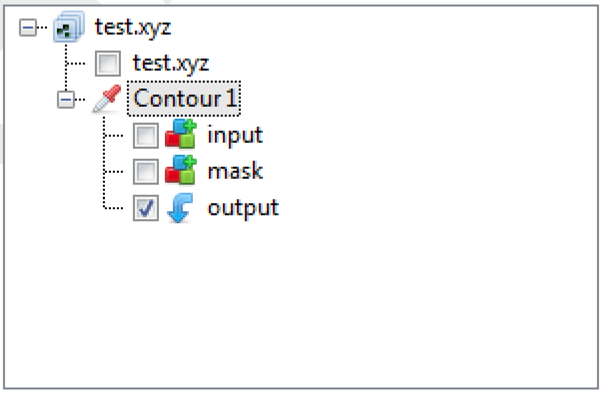


Figure 6.32: *Appearance of contour operation in the operations stack*

6.5.6 Copy to samples

This operation converts the currently selected grid coverage to a sample set, which becomes that starting point of a new subset. If polygons have been drawn, the operation will confine the copy to the interiors. The operation will not convert missing values to samples. Note that the operation does keep a reference to the original copied grid coverage; if it is changes by a re-evaluation of the stack, the changes will affect the output point cloud.

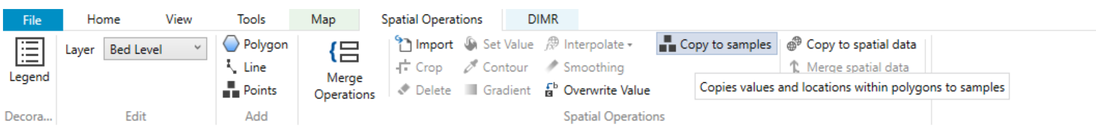


Figure 6.33: *Applying the copy to samples operation*

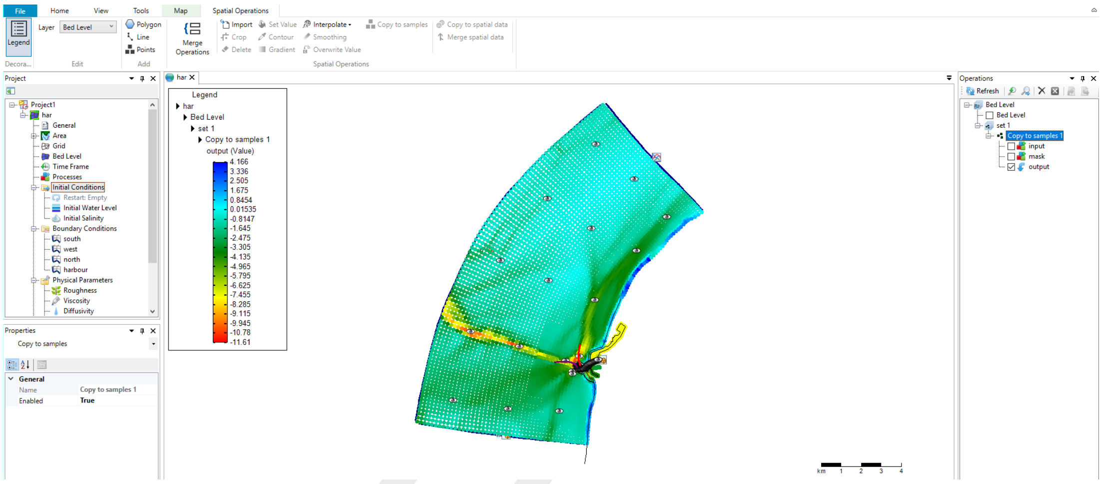


Figure 6.34: *Copy to samples operation result*

6.5.7 Copy to spatial data

This operation simply clones the grid coverage, starting a new subset with a snapshot of the currently selected operation output. Similarly to 6.5.6, it keeps a reference to the input data and will perform the clone again upon re-evaluation.

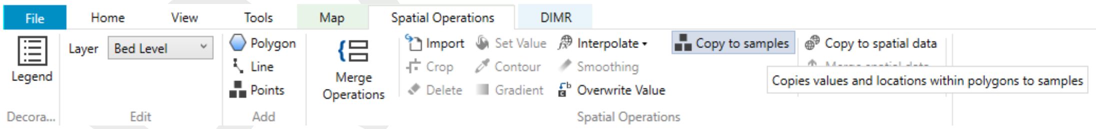


Figure 6.35: *Applying the copy spatial data operation*

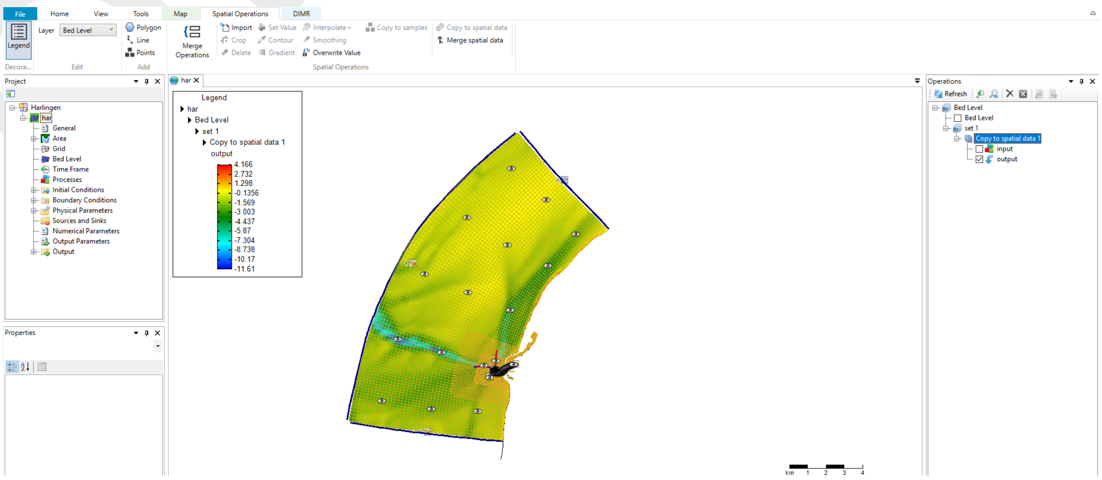


Figure 6.36: *Copy spatial data operation result*

6.5.8 Merge spatial data

Whenever a subset contains a grid coverage as its editing data (after applying e.g. 6.5.7) on the same grid as the main operation set, its result can be combined with the main set by applying this operation, similarly to the interpolation operation for sample sets, discussed below. Combining the grid coverages can be achieved with the usual point-wise methods.

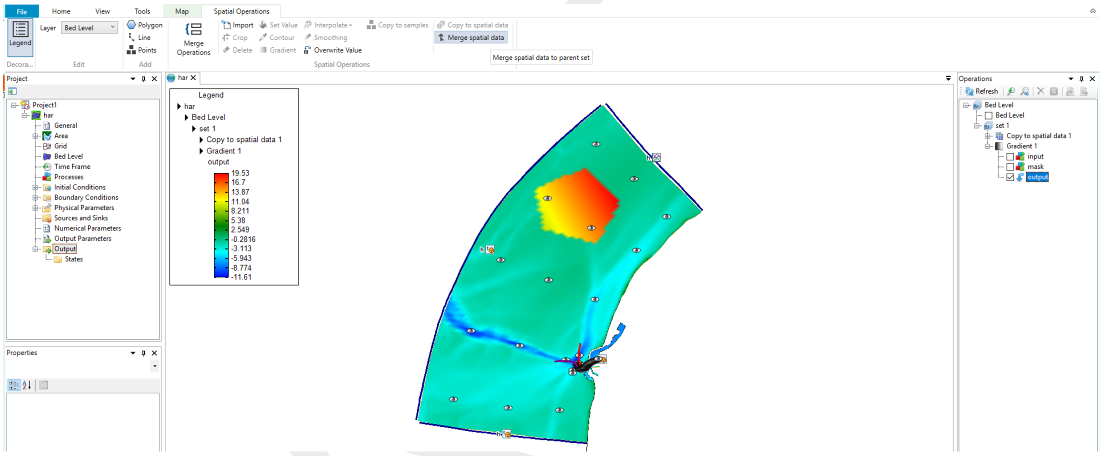


Figure 6.37: Activating the merge spatial data tool from the ribbon

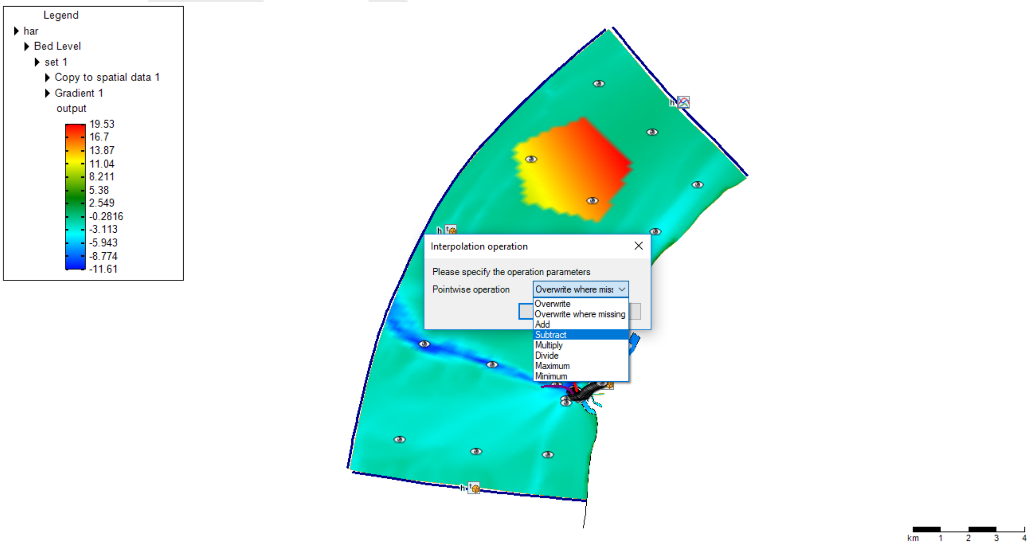


Figure 6.38: The merge operation requests a pointwise combination method

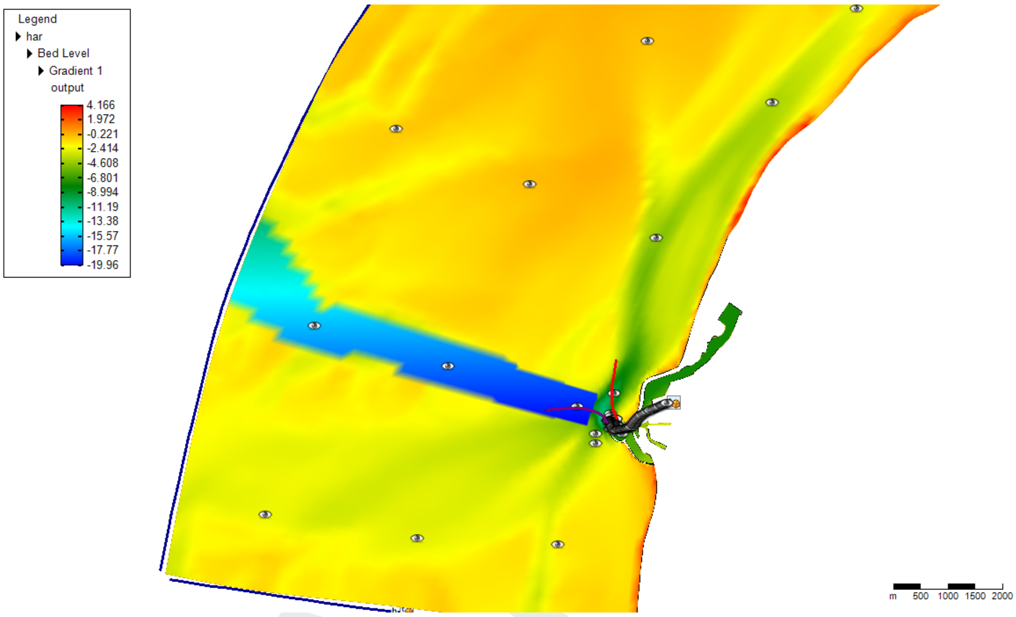


Figure 6.39: *Resulting grid coverage*

6.5.9 Gradient

The gradient operation applies a gradient to a point cloud or grid coverage (depending on which one is active). The gradient operation is activated from the 'Spatial Operations' ribbon and only available for polygon geometries or for the total data set if no polygon is provided. You have to assign the initial (start) value, the final (end) value and the going to angle (according to the Cartesian convention with 0 degrees is East, 90 degrees is North, etc **Note:** This is not working properly yet). For an example see Figure 6.40. After applying a gradient to (part of) the point cloud or coverage the operation is added to the operations stack (Figure 6.41).

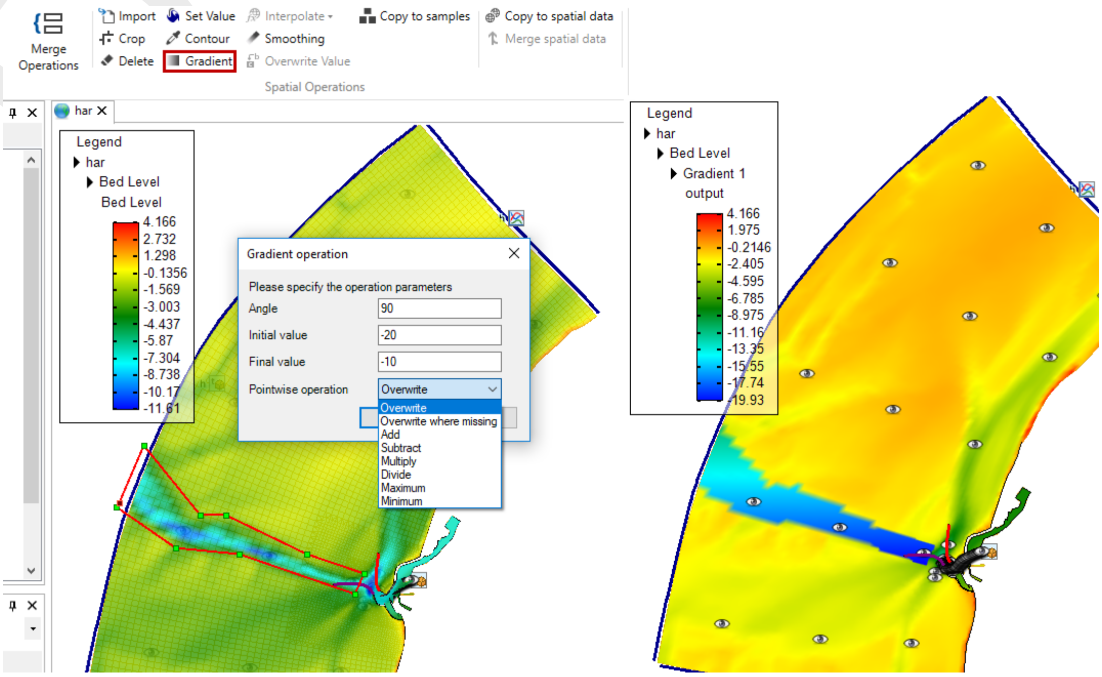


Figure 6.40: *Performing a gradient operation on a point cloud with a polygon using 'Gradient' from the 'Spatial Operations' ribbon*

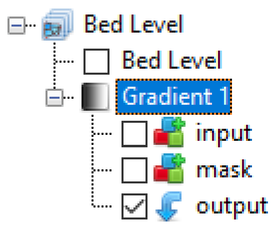


***Figure 6.41:** Appearance of gradient operation in the operations stack*

6.5.10 Interpolate

The interpolate operation is the way to get sample set(s) to a grid or network (e.g. coverage). In the operation stack this means that we are actively switching from working on a point cloud (helping to construct the coverage) to working on the selected coverage (or spatial quantity). The interpolation is performed on the data within a polygon or polygons (if provided) or all the data (if no polygons are provided). The interpolate operation can be performed on a single (selected) sample set or on multiple sample sets. Both methods are discussed below. The methods for interpolation are either

- Triangulation: performs a Delauney triangulation on the sample point set before projecting onto the grid.
- Averaging: combines sample points within a possible enlarged cell according to an algorithm of choice. The user can set the search cell expansion factor (rel. search cell size) and a threshold for the number of sample points within a cell (minimum sample points), see Figure 6.42.

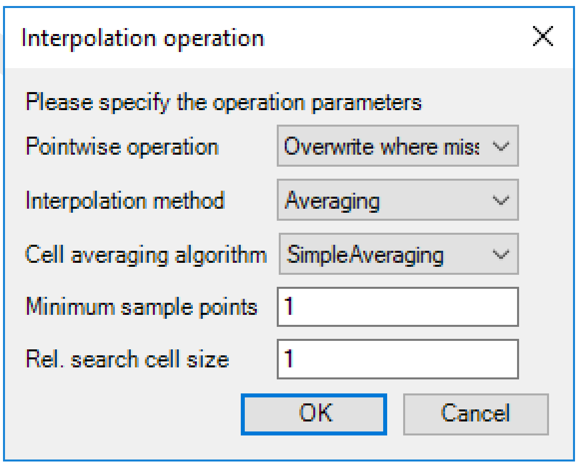


***Figure 6.42:** Interpolation Operation options*

Seven Cell averaging algorithms can be chosen by the user; see Figure 6.43. These algorithms are explained below:

- SimpleAveraging: the arithmetic mean[1] of the samples inside the search area is taken.
- ClosestPoint: the value of the closest sample inside the search area is taken.
- MaximumValue: the maximum value of the samples inside the search area is taken.

[1] The arithmetic mean is given by:
$u(\mathbf{x}) = \frac{1}{N}\sum_{k=1}^{N} u_i$

- ◇ MinimumValue: the minimum value of the samples inside the search area is taken.
- ◇ InverseWeightedDistance: the inverse-distance-weighted mean[2] of the samples inside the search area is taken. Contrary to the arithmetic mean, in which all samples contribute equally to the mean, when using the inverse-distance-weighted average, the samples close to the query point have a larger contribution to the mean.
- ◇ MinAbs: the minimum of the absolute values of the samples inside the search area is taken.
- ◇ KdTree: This is an obsolete option, which will be removed in a future release.

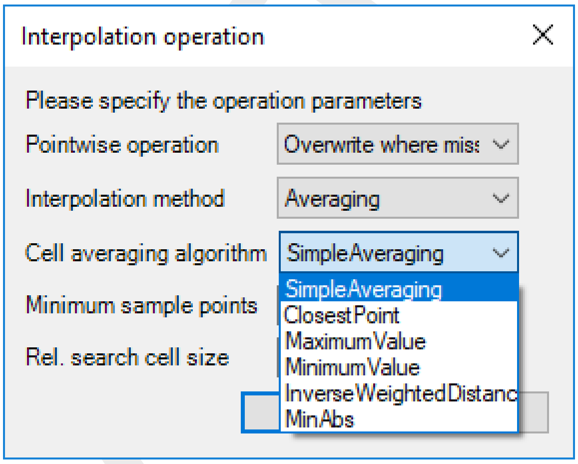


Figure 6.43: *Averaging options*

By default, the interpolation will only overwrite missing values in the gridded data set. However, if the grid coverage already contains values, the user may choose to overwrite or combine the data by a pointwise arithmetic operation.

Interpolate single (selected) set

To perform interpolation on a single sample set, select the sample set (i.e. 'set1') in the operation stack and press 'Interpolate' in the 'Map' ribbon (Figure 6.44). Since no polygon is provided in this example, all the samples will be interpolated to the grid. Use polygons if you would like to have more control over the interpolation. After the interpolation the operation is added to the operations stack (Figure 6.45). Please note that after performing the interpolation the workflow in the stack is shifting from the sample set (i.e. 'set1' - which was a side step to construct the coverage) to the coverage (i.e. 'bed level').

[2]The inverse-distance-weighted mean is given by:

$$u(\mathbf{x}) = \begin{cases} \dfrac{\sum_{i=1}^{N} w_i(\mathbf{x}) u_i}{\sum_{i=1}^{N} w_i(\mathbf{x})}, & \text{if } d(\mathbf{x},\mathbf{x}_i) \neq 0 \text{ for all } i, \\ u_i, & \text{if } d(\mathbf{x},\mathbf{x}_i) = 0 \text{ for some } i, \end{cases}$$

$$w_i(\mathbf{x}) = \frac{1}{d(\mathbf{x},\mathbf{x}_i)}$$

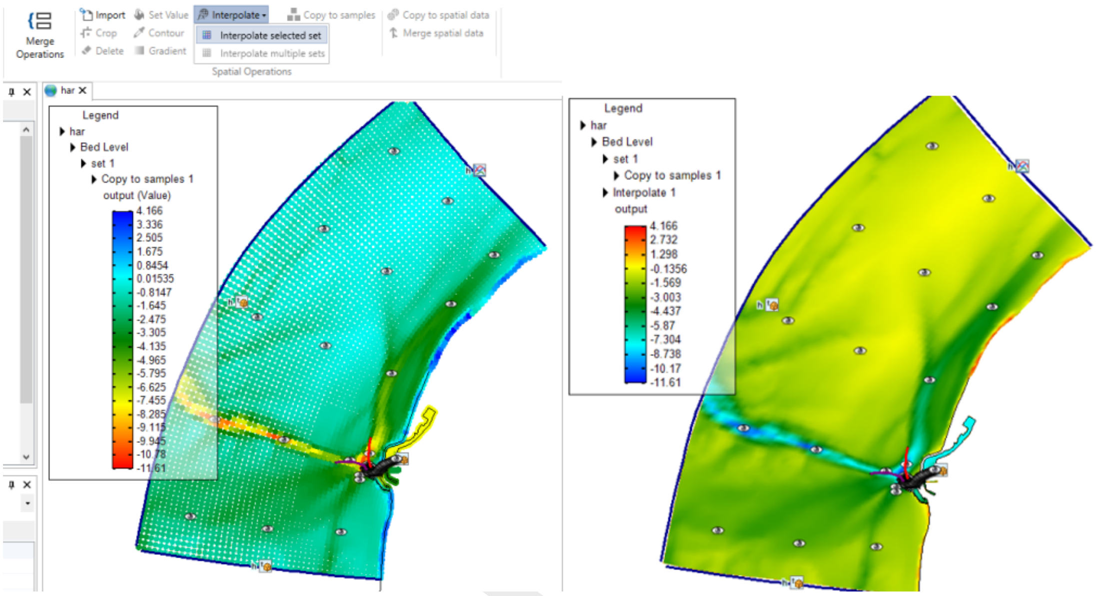


Figure 6.44: Performing an interpolation operation on a single sample set (without using a polygon) using 'Interpolate' from the 'Spatial Operations' ribbon

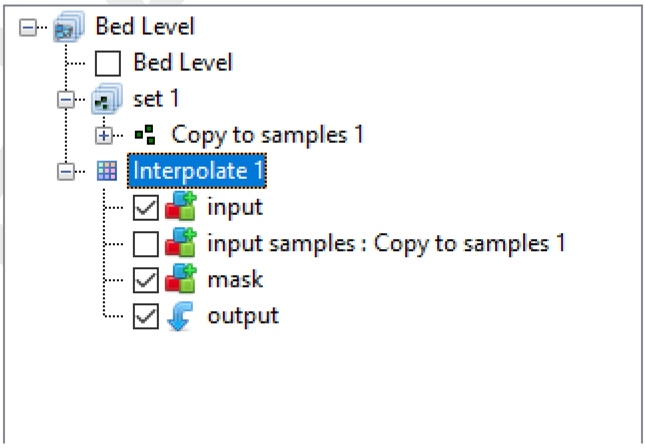


Figure 6.45: Appearance of interpolation of 'set1' to the coverage 'bed level' in the operations stack

Interpolate multiple sets

To perform interpolation on multiple sample sets, select the active coverage (i.e. 'bed level') in the operation stack and press 'Interpolate' in the 'Map' ribbon (Figure 6.46). In the popup you can select which sample sets to include in the interpolation (in this example both). Since no polygon is provided, all the samples (from the two sets) will be interpolated to the grid. Use polygons if you would like to have more control over the interpolation. After the interpolation the operation is added to the operations stack (Figure 6.47). Again note that after performing the interpolation the workflow in the stack is shifting from the sample set (which was a side path to construct the coverage) to the coverage (i.e. bed level).

Note: Please note that interpolation of multiple sample sets can also be achieved by importing/combining different sample sets into the same set in the stack instead of using two separate sets. In this case you can just interpolate the single (selected) set.

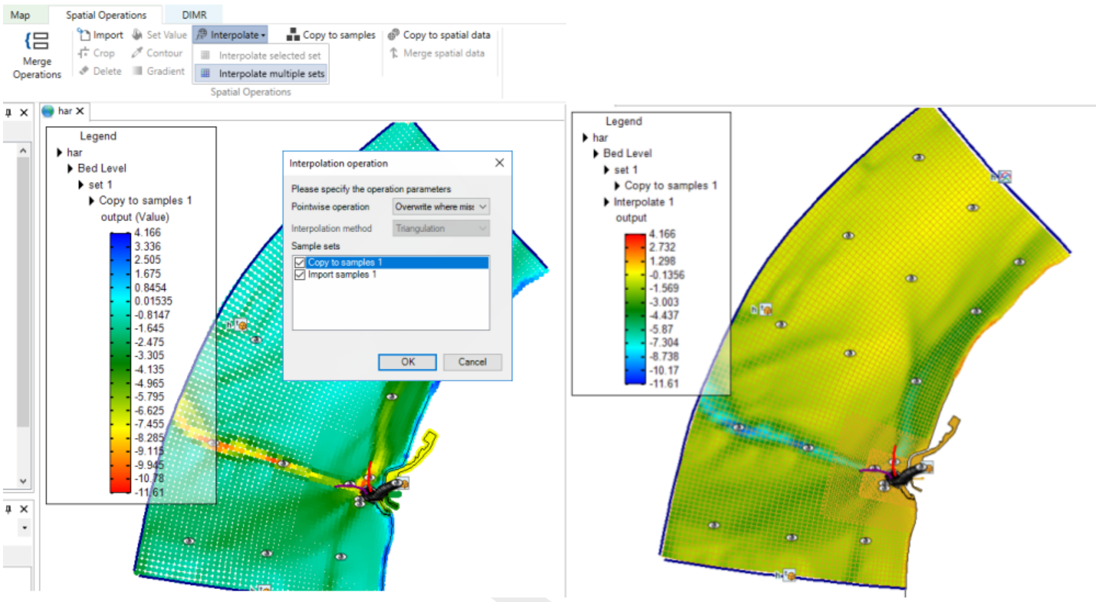


Figure 6.46: Performing an interpolation operation on multiple sample sets (without using a polygon) using 'Interpolate' from the 'Spatial Operations' ribbon

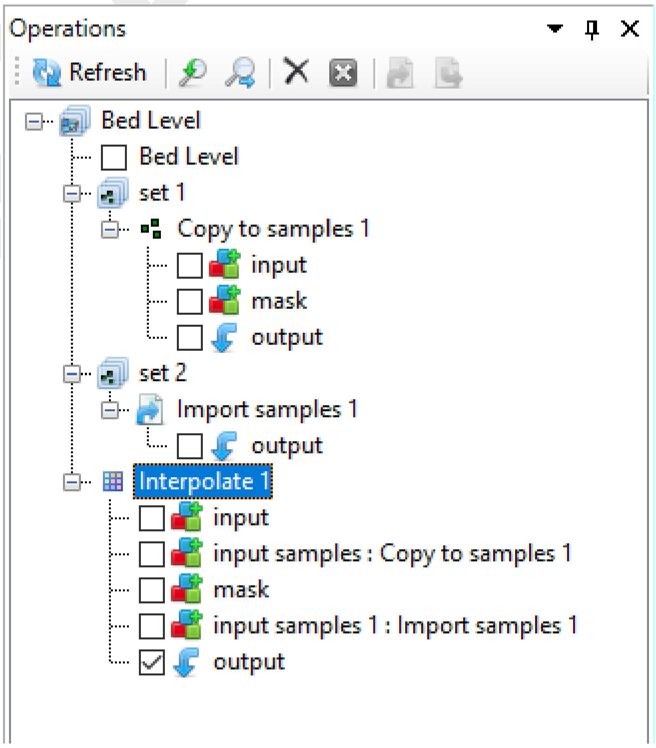


Figure 6.47: Appearance of interpolation of 'set1' and 'set2' to the coverage 'bed level' in the operations stack

6.5.11 Smoothing

The smoothing operation smooths out (steep) gradients in a point cloud or coverage (depending on which one is active). The smoothing operation is activated from the 'Spatial Operations' ribbon and only available for polygon geometries or for the total data set if no polygon is provided. You have to assign the smoothing exponent and number of smoothing steps. The higher the exponent and the number of smoothing steps, the heavier the smoothing. For an example see Figure 6.48. After applying smoothing to (part of) the point cloud or coverage the operation is added to the operations stack (Figure 6.49).

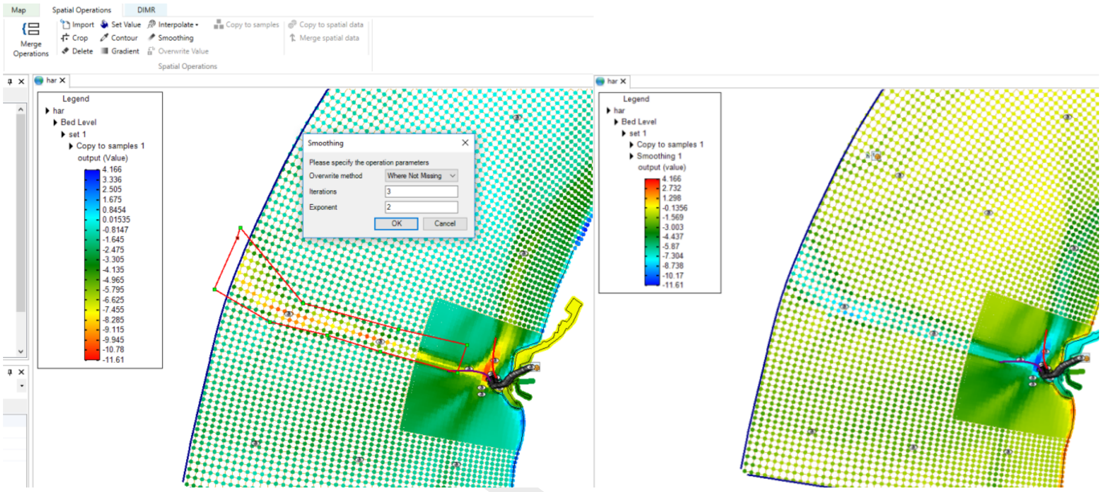


Figure 6.48: *Performing a smoothing operation on a point cloud with a polygon using 'Smoothing' from the 'Spatial Operations' ribbon*

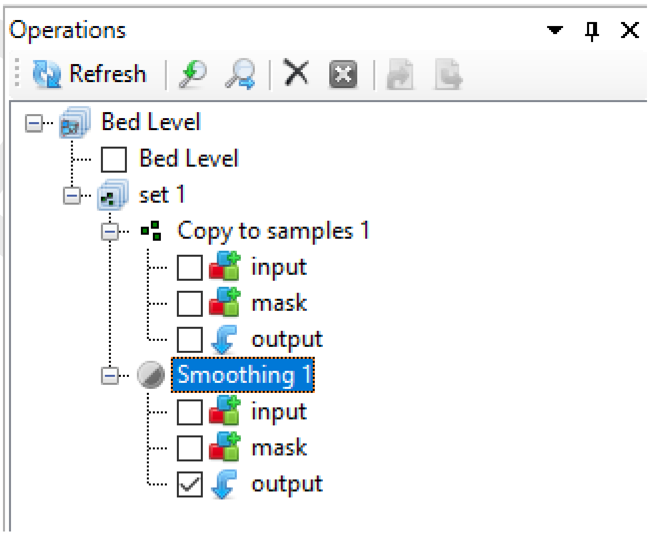


Figure 6.49: *Appearance of smoothing operation in the operations stack*

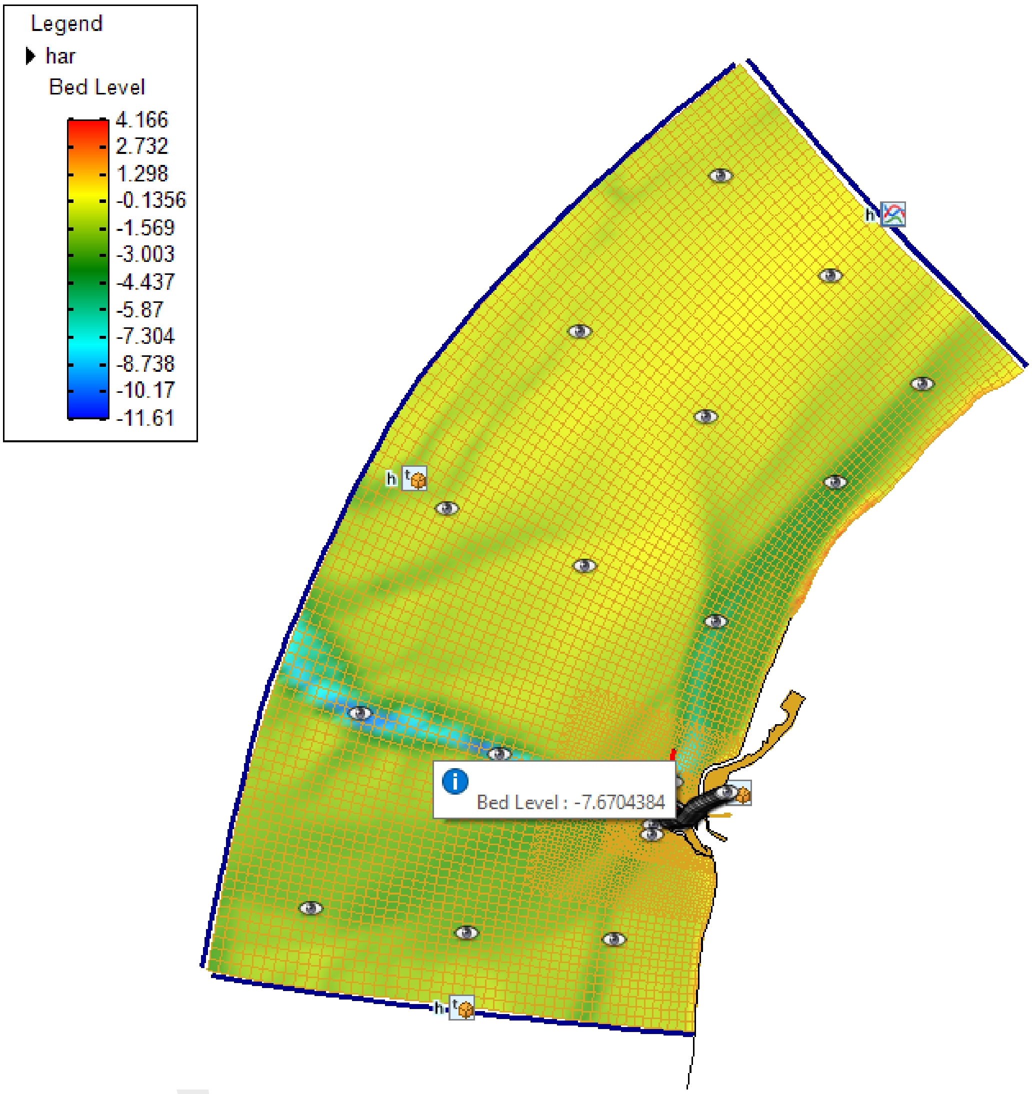


Figure 6.50: *The cursor for the overwrite operation showing the value of the closest coverage point*

6.5.12 Overwrite (single) value

The ‘overwrite (single) value’ operation allows you to edit single values on the active coverage after the interpolation. The ‘overwrite (single) value’ operation is activated from the ‘Spatial Operations’ ribbon. There is no geometry required for this operation. Upon selecting the operation from the ribbon a cursor will become active showing the coverage value closest to the cursor in a tooltipstring (Figure 6.50). Upon clicking LMB a popup appears in which you can overwrite the value of this coverage point Figure 6.51. After applying the overwrite operation it is added to the operations stack (Figure 6.52).

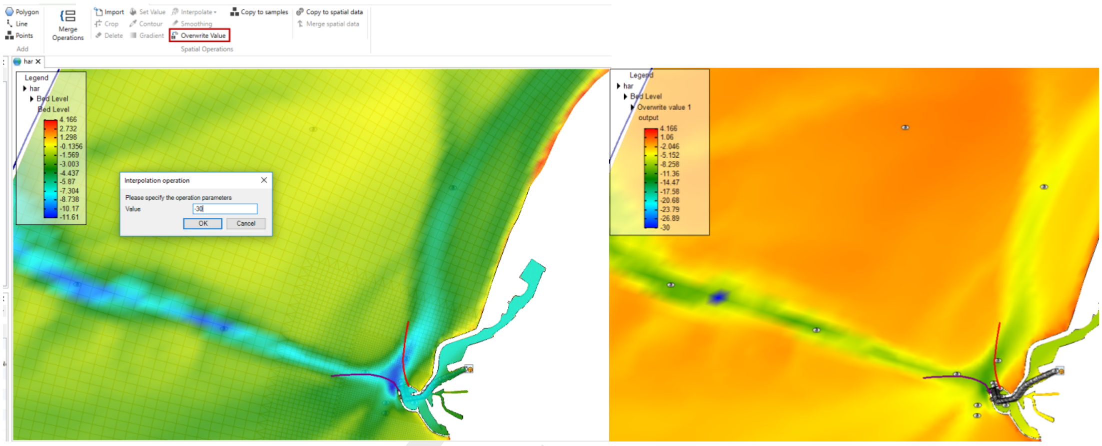


Figure 6.51: *Performing an overwrite operation on a coverage point using 'Single Value' from the 'Spatial Operations' ribbon*

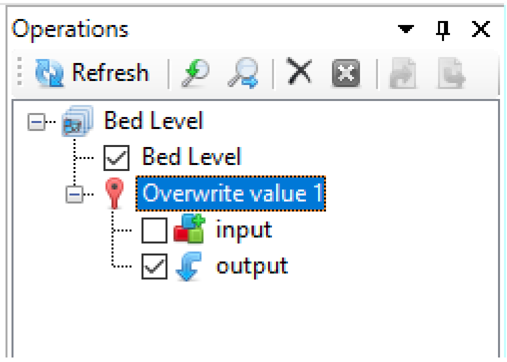


Figure 6.52: *Appearance of overwrite operation in the operations stack*

6.6 Operation stack

The operation stack keeps track of the workflow of spatial operations that you performed. This helps you to make transparent how you arrived at your 'final' dataset without having to save all the intermediate datasets (steps) separately. Moreover, the stack is reproducible and easily editable without having to start all over again. This section describes the stack workflow (section 6.6.1), how to edit operation properties (section 6.6.2), how to enable/disable (section 6.6.3), delete (section 6.6.4), refresh(section 6.6.5) operations, quick links (section 6.6.6) and import/export functionality (section 6.6.7).

Note: Currently, the stack is saved in the User Interface project upon saving the project. The next time you open the project, the stack will reappear. The stack is not (yet) saved in a human readable/editable file.

6.6.1 Stack workflow

Upon performing a spatial operation, the ‘Operations’ panel will open (see Figure 6.53) with the operations stack (tree). The stack first shows on which point cloud or coverage you are working (in this example ‘bed level’). Subsequently, all the operations on this dataset are listed. For each operation you can inspect what the input, mask (e.g. the geometry used for the operation) and output are for the operation (Figure 6.54). By default the stack continues from the last operation that you performed. If you wish to work on a different dataset or operation within a dataset, you have to select that dataset or operation in the ‘Operations’ panel with the LMB.

When working on a coverage, point clouds (or sets) can be used to construct the coverage. In that case the stack jumps from the ‘trunk’ to a ‘branch’ and the subsequent operations are performed on the point cloud (see Figure 6.53). By selecting the set or coverage in the ‘Operations’ panel you determine on which dataset you are working. The interpolate operation (section 6.5.10) allows you to bring data from the point cloud (branch) to the coverage (trunk). See also Figure 6.53.

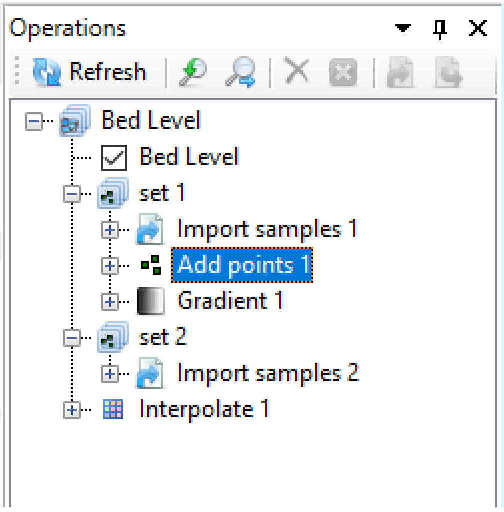


Figure 6.53: *The ‘Operations’ panel with the operations stack. In this example ‘bed level’ is the coverage (e.g. trunk) that is edited. The point clouds ‘set 1’ and ‘set 2’ (e.g. branches) are used to construct the ‘bed level’ coverage.*

6.6.2 Edit operation properties

For each operation that you performed the properties (such as the value or ‘Pointwise operation’ of a ‘Set Value’ operation) can be edited using the ‘Properties’ window (Figure 6.55). **Note:** Please note that the mask of an operation cannot (yet) be edited. By editing the operation properties the operation stack becomes ‘out of sync’ and has to be refreshed for the changes to become active (see section 6.6.5).

6.6.3 Enable/disable operations

You can (temporarily) enable/disable operations by selecting the operation and pressing boxed cross icon in the stack menu (Figure 6.56). Upon disabling an operation the operation will be made grey in the stack and the operation is not taken into account anymore upon evaluation of the overall result. The result of disabling an operation is not directly activated. This is indicated in the stack with the ’out of sync’ exlamation mark (Figure 6.56). You need to refresh the stack (see section 6.6.5) for the changes to become active.

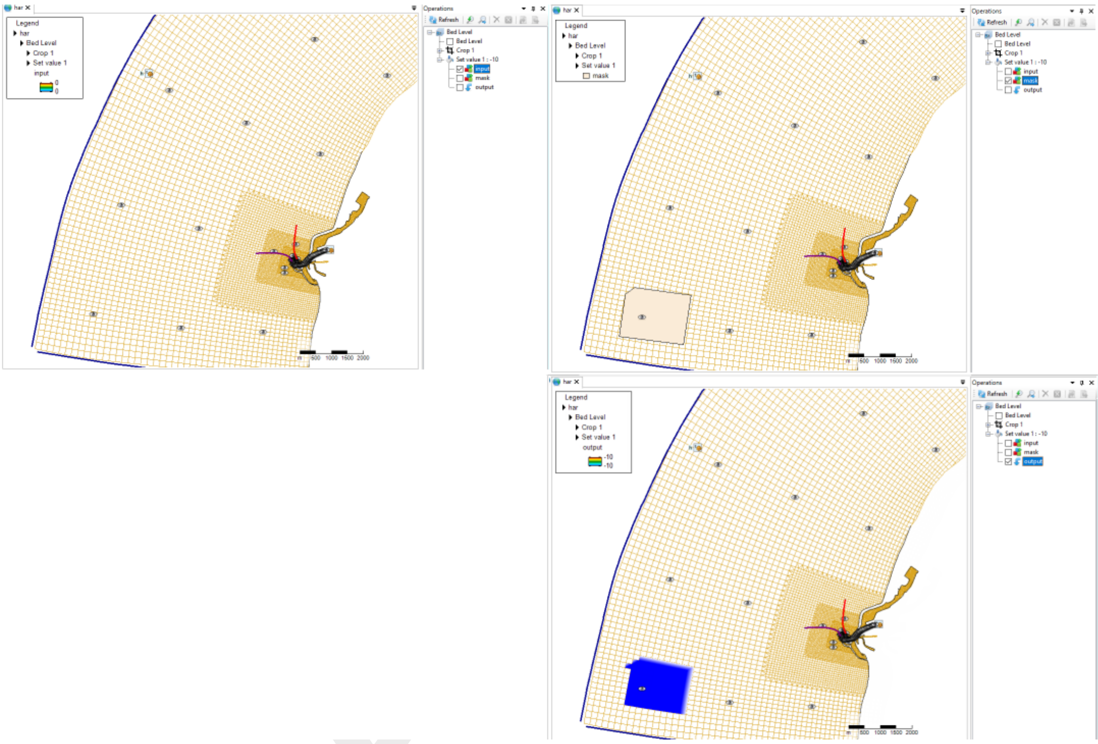


Figure 6.54: *Input for the operation (top panel), mask for the operation (middle panel) and output of the operation (bottom panel)*

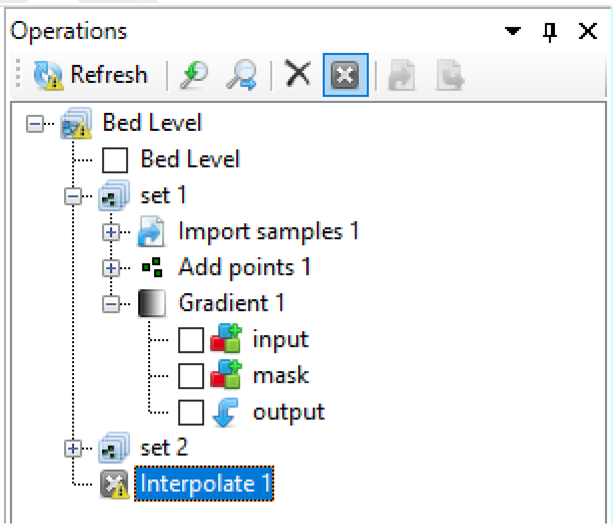


Figure 6.56: *Disabling an operation using the boxed cross icon in the stack menu. The operation will become grey. Note the exlamation marks marking the stack 'out of sync'.*

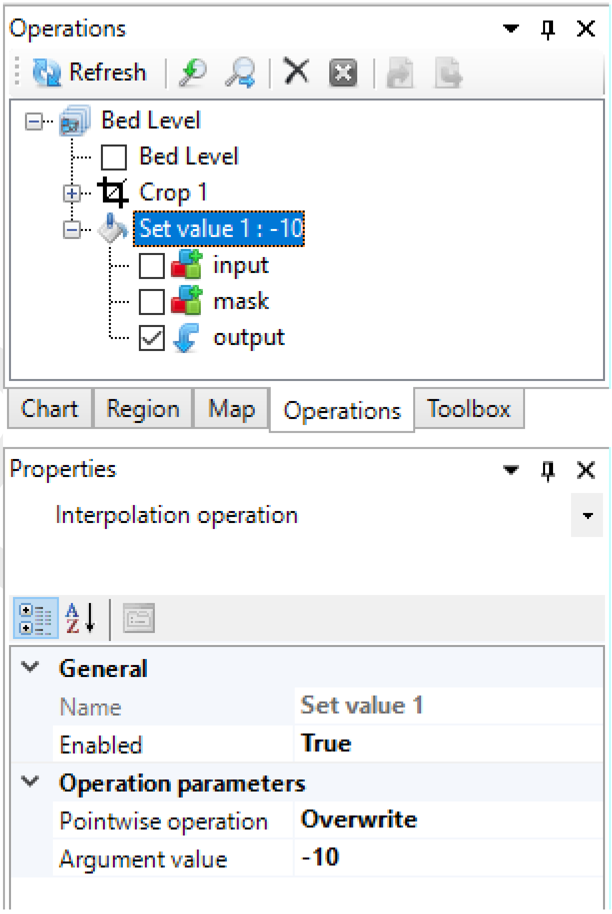


Figure 6.55: *Editing the value or 'Pointwise operation' of a 'Set Value' operation using the properties panel*

6.6.4 Delete operations

To delete an operation permanently you have to select the operation and either press the cross icon (Figure 6.57) or use the context menu and select delete (Figure 6.58). The operation will be removed from the stack. The result of deleting an operation is not directly activated. This is indicated in the stack with the 'out of sync' exlamation mark. You need to refresh the stack (see section 6.6.5) for the changes to become active.

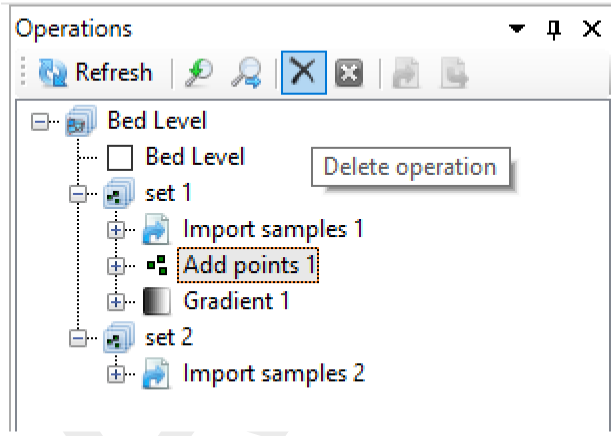


Figure 6.57: *Removing an operation from the stack using the cross icon in the stack menu*

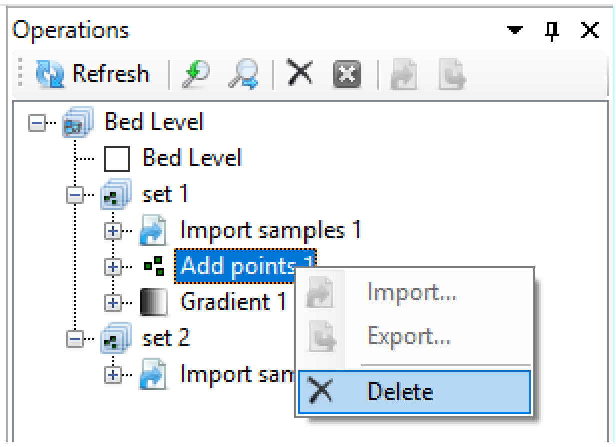


Figure 6.58: *Removing an operation from the stack using the context menu on the selected operation*

6.6.5 Refresh stack

When the stack is marked 'out of sync' by exclamation marks, you can refresh the stack by pressing the 'Refresh' button for the changes to become active (Figure 6.59). Upon refreshing the stack all the (enabled) operations in the stack will be (re-)evaluated. **Note:** Please note that refreshing the stack can take some time when large datasets are (re-)evaluated!

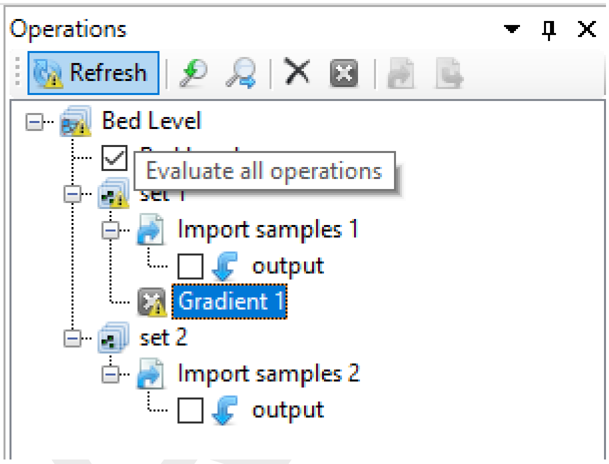


Figure 6.59: *Refresh the stack using the 'Refresh' button so that all operation are (re-)evaluated*

6.6.6 Quick links

The stack menu contains two quick links to quickly show the original dataset (e.g. where you started from, Figure 6.60) and the end result of the spatial operations (Figure 6.61).

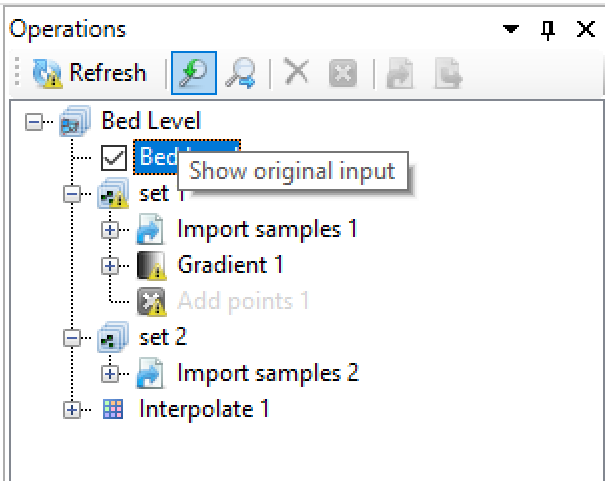


Figure 6.60: *Quick link to the original dataset before performing any spatial operations*

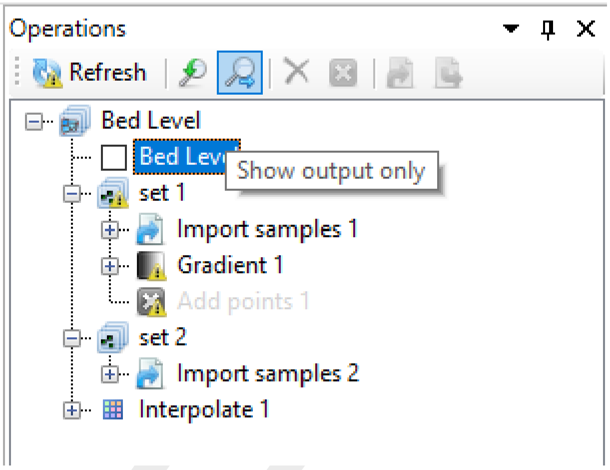


Figure 6.61: *Quick link to the output after performing all (enabled) operations*

6.6.7 Import/export

Note: Importing and exporting data into or from the stack is still under construction

7 Model coupling

7.1 Model coupling using integrated models within the User Interface

Besides adding stand-alone models to a project, it is also possible to create integrated models or upgrade stand-alone models to integrated models. A flow model and a rainfall-runoff model can be independent, but they may also be combined in an integrated model. An integrated model consists of a set of models which may run sequentially or in parallel. The first means that the output of the first sub-model is generated and used as input for the second model within the integrated model without any feedback from the second model to the first. The second means that the output of one sub-model is used as input of the second sub-model within the integrated model and vice versa. In this way the models are coupled externally (Morita and Yen, 2002) on a time step basis.

An example of an integrated model is a D-RTC model (a Real Time Control model) combined with a D-Flow FM model (Figure 7.1). When the integrated model is run, each timestep both the D-Flow FM model and the D-RTC model are run sequential or parallel. The output of the D-Flow FM model is input for the D-RTC model. In case the models are run parallel the in- and output are exchanged every time step: the output of the D-RTC model in the previous timestep is used as input for the D-Flow FM model, and vice-versa. The models are coupled on the structures which are controlled.

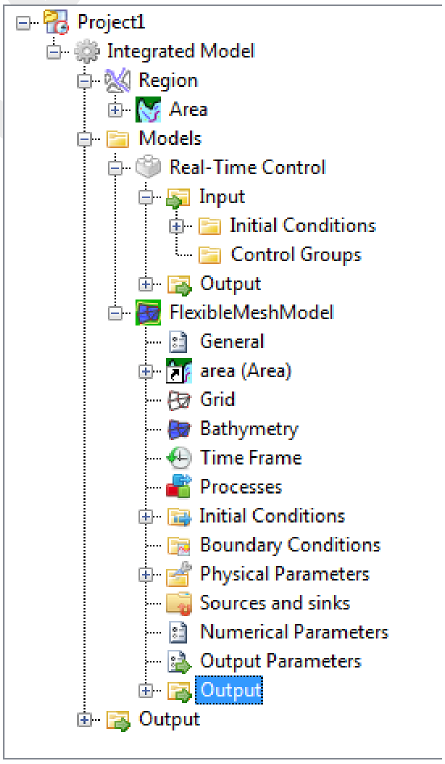


Figure 7.1: *Example of an integrated model*

7.2 Model coupling using the BMI standard

Most computational cores available within the User Interface can not only be coupled using the integrated model functionality within the D-Flow FM GUI, but can also be coupled outside the Delta Shell GUI for running on clusters. For such model runs, which usually require a high performance, the models are coupled using the Basic Model Interface (BMI). For a description of the BMI the user is referred to http://csdms.colorado.edu/wiki/BMI_Description.

7.3 Model coupling using the OpenMI standard

Applications that are based on the Delta Shell framework, like SOBEK 3 and Delft 3D Flexible Mesh (FM), are so-called OpenMI-compliant with version 1.4 and 2.0 of the OpenMI-standard. The OpenMI-standard, the documentation and the supporting tools can be found at http://www.openmi.org.

SOBEK 3- and Delft 3D FM-models can be used in an OpenMI-configuration by providing a so-called <∗.omi>-file in which the project (<∗.dsproj>) is specified. An example follows here.

```
<?xml version="1.0"?>
<LinkableComponent xmlns="http://www.openmi.org"
      Type="DeltaShell.OpenMIWrapper.DeltaShellOpenMILinkableComponent"
            Assembly="../../bin_sobek3_openmi/DeltaShell.OpenMIWrapper.dll">
      <Arguments>
            <Argument Key="DsProjFilePath" ReadOnly="true" Value="../../test_models/boezem.dsp
            <Argument Key="DsProjModelName" ReadOnly="true" Value="integrated model" />
            <Argument Key="ModelId" ReadOnly="true" Value="boezem" />
            <Argument Key="ResultingDsProjFilePath" ReadOnly="true" Value="./boezem_out.dsproj
            <Argument Key="SplitSpecificElementSets" ReadOnly="true" Value="grid_point;reach_s
            <Argument Key="SeparateProcess" ReadOnly="true" Value="true" />
      </Arguments>
</LinkableComponent>
```

8 Command-line and scripting

8.1 Introduction

Setting up and running models using (ASCII) scripts can be a very useful work flow for modellers that would like to perform batch studies where a single model parameter needs to be changed, to perform batches of calculations on large computational clusters, or for simply reproducing the model setup procedure (before extending the model to a more complex version for example) over and over again. To support the here-fore mentioned work flows, the Delta Shell framework supports the IronPython scripting language encapsulated within the Delta Shell scripting toolbox. It may be used from within the Delta Shell GUI (only on Windows machines), as well as from a console/terminal on any platform. In the following sections, the use of the Delta Shell command-line and scripting functionality will be explained based on the use of existing D-Flow 1D projects.

8.2 Scripting editor overview

The scripting editor within Delta Shell contains the following features:

- ◇ Syntax highlighting section 8.2.1
- ◇ Code completion section 8.2.2
- ◇ Show extra characters section 8.2.3
- ◇ Regions section 8.2.4
- ◇ Local variables section 8.2.5
- ◇ Ribbon section 8.2.7
- ◇ Shortcuts section 8.2.8

8.2.1 Syntax highlighting

With syntax highlighting all the words and types that are known in Python will be shown as colored text. To configure the color schema go to the Delta Shell installation folder and navigate to the plugins/Delta Shell.Plugins.Scripting.Gui folder. Here the XML-file <Python.xshd> contains all the information used for syntax highlighting. An example is presented in Figure 8.1.

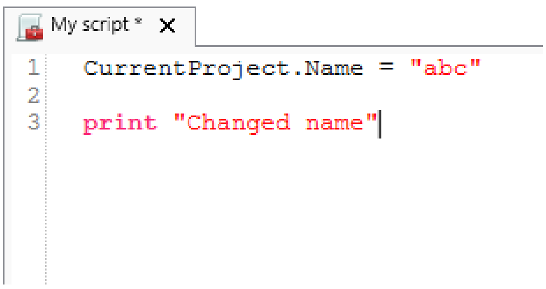


Figure 8.1: *Example of syntax highlighting within the Delta Shell scripting editor.*

8.2.2 Code completion

Code completion is used to give the you quick access to all properties, events and methods of the variable that you are working on. When activated a list will be shown showing all the options for this variable and the documentation for the selected element. To filter the list continue typing, and the list will filter out all non matching elements. To select an element press the up and down keys and enter to confirm. If an empty list is shown, then the type of the variable can not be determined or the variable has no elements to show. Code completion is activated when a *'.'* is typed or when *Ctrl + Space* is pressed. An example is shown in Figure 8.2.

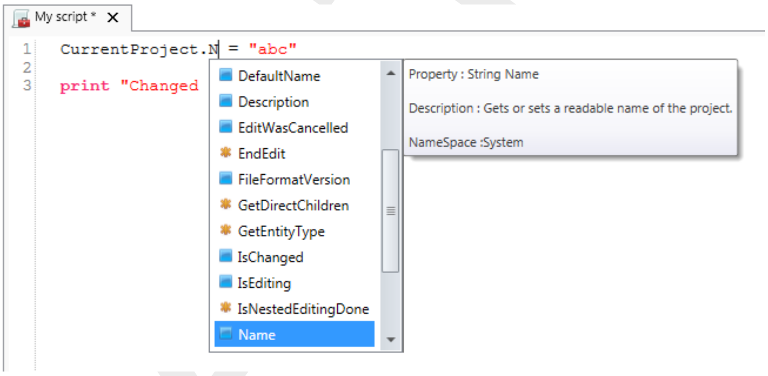


***Figure 8.2:** Example of code completion within the Delta Shell scripting editor.*

8.2.3 Show extra characters

These options are added to show you the space, tab and line end characters. This can be important because pythons logic uses indenting for its statements. Figure 8.3 to Figure 8.5 show examples of these extra characters.

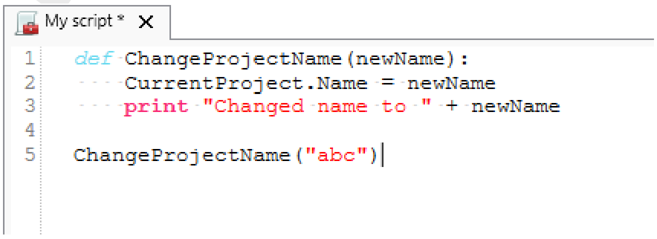


***Figure 8.3:** Showing spaces within the Delta Shell scripting editor.*

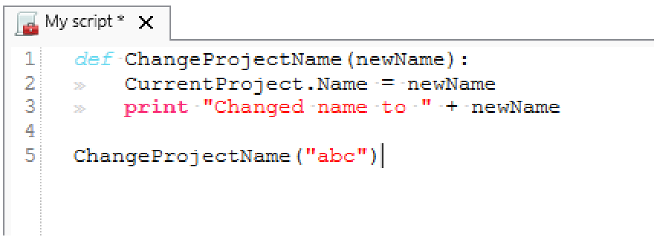


***Figure 8.4:** Showing tabs within the Delta Shell scripting editor.*

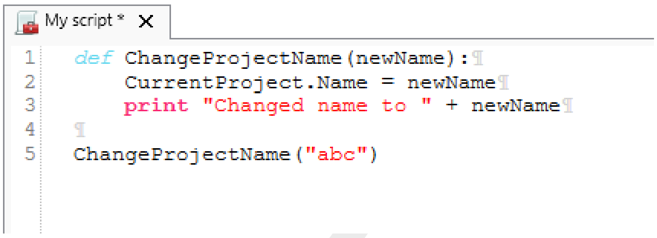


Figure 8.5: *Showing EOL characters within the Delta Shell scripting editor.*

8.2.4 Regions

Regions are used to mark code blocks, and gives you the ability to collapse these blocks to a single line with a title. Figure 8.6 and Figure 8.7 shows an examples of such a region.

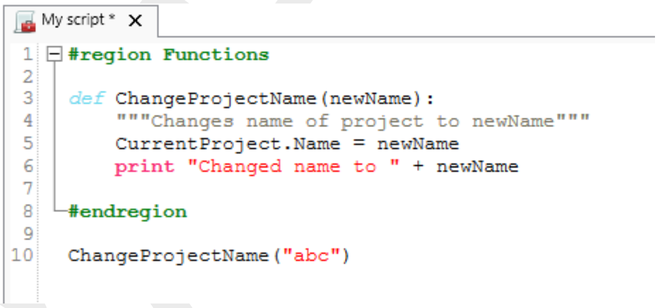


Figure 8.6: *Open region within the Delta Shell scripting editor.*

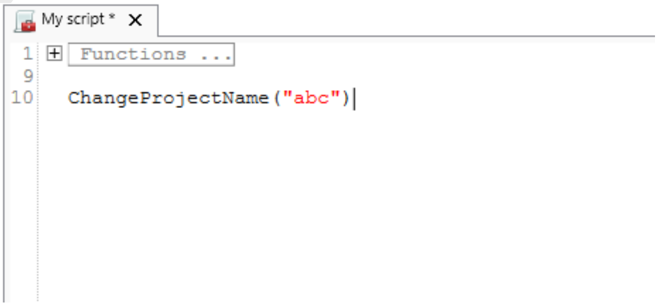


Figure 8.7: *Closed region within the Delta Shell scripting editor.*

8.2.5 Local variables

The local variable option gives you an overview of the declared variables in your script. The variables are declared when you run the part of your script that declares the variables. It will show the name of the variable followed by a string representation of the variable value and the type. An example is presented in Figure 8.8. To get more detail about the variable, select it and the property window will show you all the details about the variable as shown in Figure 8.9.

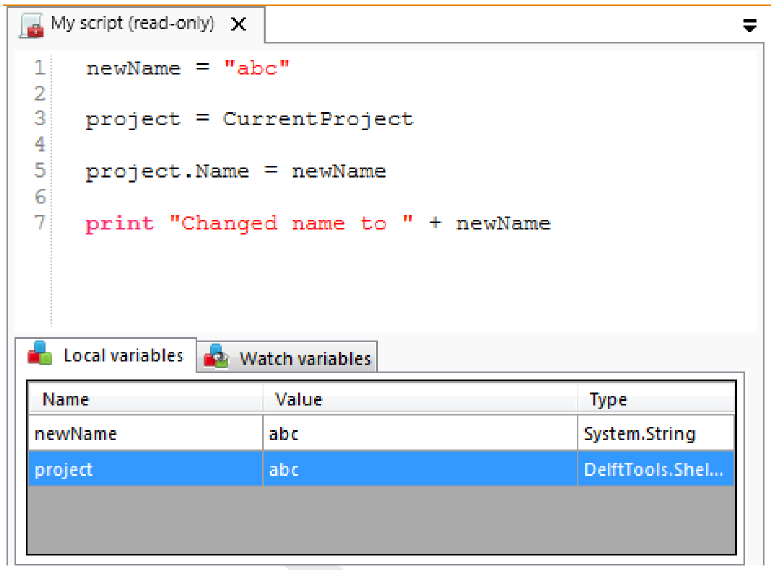


Figure 8.8: *Local variables within the Delta Shell scripting editor.*

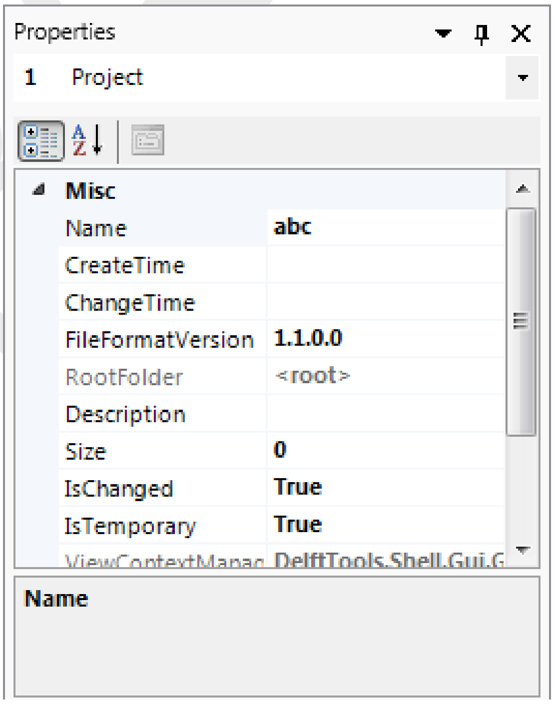


Figure 8.9: *Properties of variables within the Delta Shell scripting editor.*

There is also an option called watch variables that allows you to add your own variables that you want to observe. This works exactly the same as the local variables, with one exception. The variables are not automatically reloaded after running a part of the script. Reloading can be done by pressing the *refresh values* button below the watch variables. An example of a watch variable is shown in Figure 8.10.

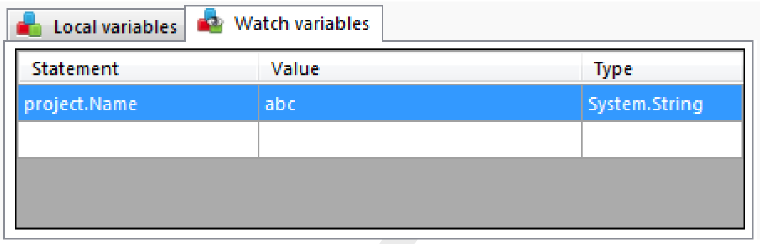


Figure 8.10: *Watch variables within the Delta Shell scripting editor.*

8.2.6 Ribbon

8.2.7 Scripting

When you open the scripting editor in the User Interface, a Scripting *ribbon* category will appear. This ribbon has the following additional options (see also Figure 8.11), which are described in Table 8.1:

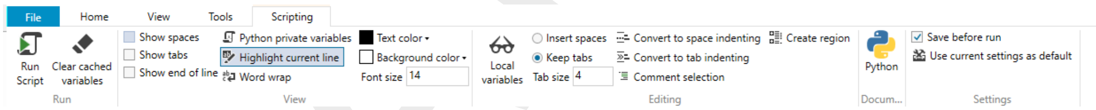


Figure 8.11: *The scripting* ribbon *within the User Interface.*

Table 8.1: *Functions and their descriptions within the scripting* ribbon *of the User Interface.*

Function	**Description**
Run script	Executes the selected text. If no text is selected then it will execute the entire script
Clear cached variables	Clears all variables and loaded libraries from memory
Debugging	Enables/Disables the debug option. When enabled you can add breakpoint to the code (using `F9` or clicking in the margin) and the code will stop at this point before executing the statement (use F10 (step over) or F11 (step into) for a more step by step approach)
Python variables	Show or hide python variables (like _var_) in code completion
Insert spaces/tabs	Determines if spaces or tab characters are added when pressing tab
Tab size	Sets the number of spaces that are considered equal to a tab character
Save before run	Saves the changes to the file before running
Create region	Creates a new region surrounding the selected text
Comment selection	Comments out the selected text
Convert to space indenting	Converts all tab characters in the script to spaces. The number of spaces is determined by Tab size
Convert to tab indenting	Converts all x number of space characters (determined by Tab size) in the script to tabs

Table 8.1: *Functions and their descriptions within the scripting* ribbon *of the User Interface.*

Function	**Description**
Python (documentation)	Opens a link to the python website, showing you the python syntax and standard libraries

8.2.8 Shortcuts

The shortcut keys of the scripting editor within the User Interface are documented in Table 8.2.

Table 8.2: *Shortcut keys within the scripting editor of the User Interface.*

Shortcut	**Function**
`Ctrl + Enter`	Run selection (or entire script with no selection)
`Ctrl + Shift + Enter`	Run current region (region where the cursor is in)
`Ctrl + X`	Cut selection
`Ctrl + C`	Copy selection
`Ctrl + V`	Paste selection
`Ctrl + S`	Save script
`Ctrl + -`	Collapse all regions
`Ctrl + +`	Expand all regions
`Ctrl + "`	Comment or Uncomment current selection
`Ctrl + W`	Add selection as watch variable
`Ctrl + H`	Highlight current selection in script (press esc to cancel)
`F9`	Add/remove breakpoint (In debug mode only)
`F5`	Continue running (In debug mode only — when on breakpoint)
`Shift + F5`	Stop running (In debug mode only — when on breakpoint)
`F10`	Step over current line and break on next line (In debug mode only - when on breakpoint)
`F11`	Step into current line if possible, otherwise go to next line (In debug mode only — when on breakpoint). This is used to debug functions declared in the same script (that have already runned)

8.3 Toolbox overview

The toolbox in Delta Shell consists of three folders, see Figure 8.12. The first two folders "Models" and "Items" contain lists of models and items that can be added to your project. The third folder "Scripts" shows all the sub-folders and (python) script files that are in the scripting folder.

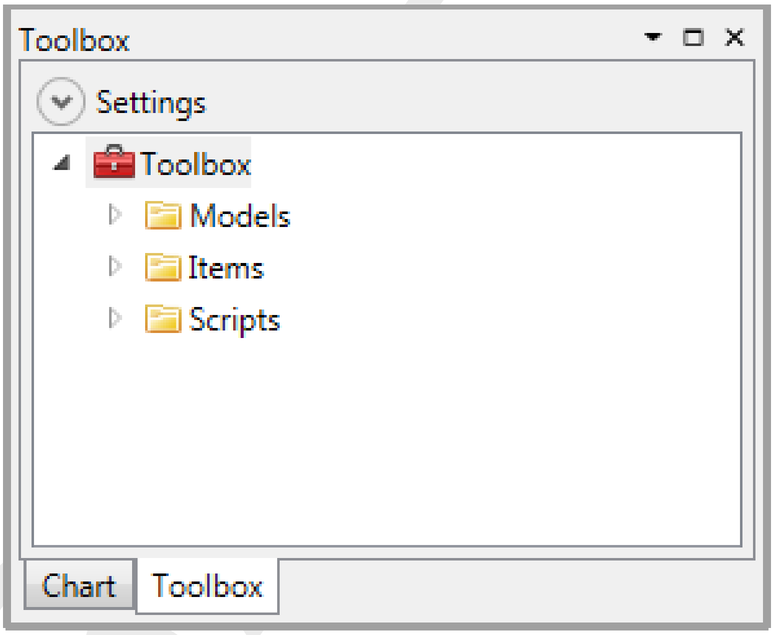


Figure 8.12: *The* ***Toolbox*** *window within Delta Shell.*

The scripting folder is a folder on your file system that is used for storing scripts that you would want to share between projects. This folder is synchronized with the file system so that you can manage the scripts in the toolbox and with your file browser (windows explorer). To change the location of the scripting folder right click on the toolbox node and select *Change scripting folder*.

8.3.1 Exploring the scripting folder

The toolbox gives you an easy way to work and manage the scripts in the scripting folder. Figure 8.13 shows an expanded view of the scripting folder. By using the + sign next to the file you can see all the declared functions and classes that are in that script. By selecting the script or one of the functions you can also see the documentation of the selection in the property grid. By double clicking one of the functions you will open the scripting editor at the start of the declaration of the function.

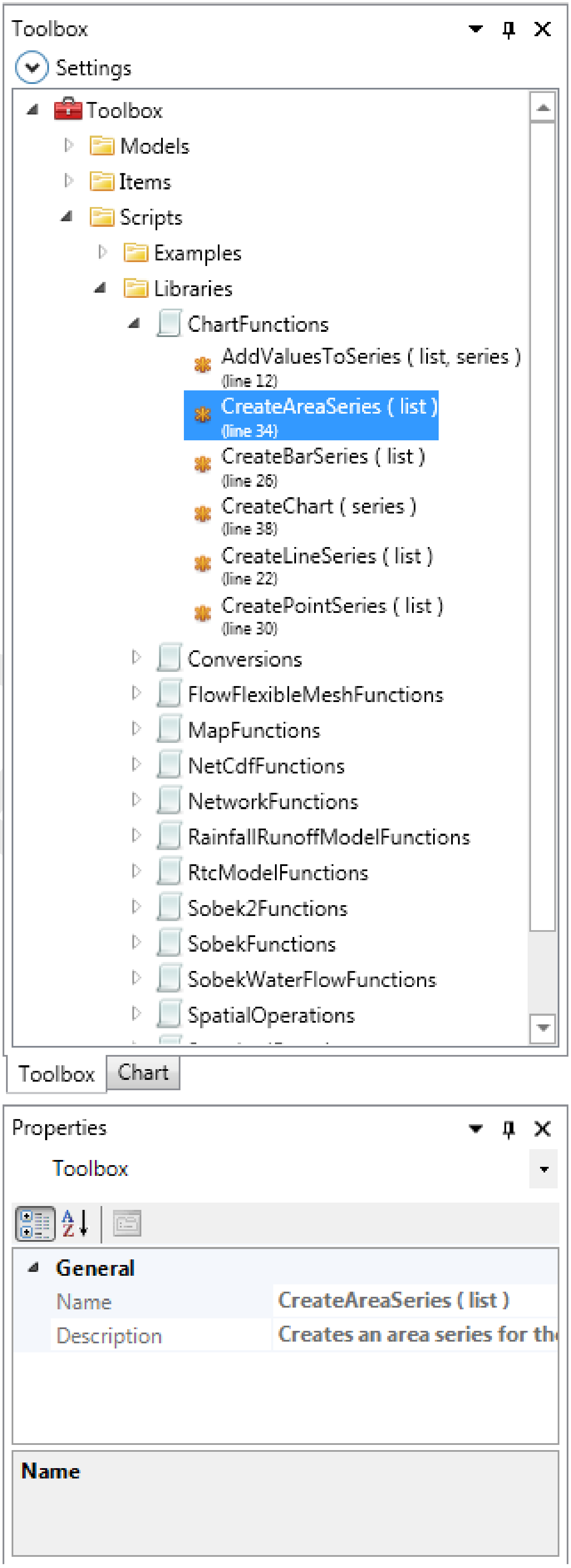


Figure 8.13: *Expanded view of the Scripts folder.*

8.3.2 Working with scripts

After selecting a script in the toolbox and right clicking it, you get a context menu with the options as shown in Figure 8.14. The options are described in Table 8.3.

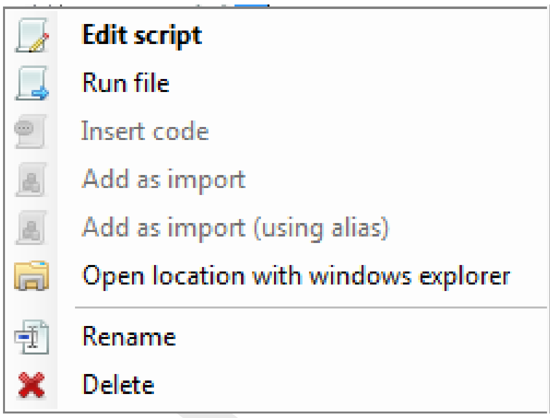


Figure 8.14: *Context menu of a script item within the toolbox.*

Function	Description
Edit script	Opens the scripting editor for this script
Run file	Runs the whole script without opening the editor
Insert code	Inserts the code of the selected script into the current script (the script open in the scripting editor)
Add as import	Adds the selected script as import to the current script (the script open in the scripting editor)
Open location with windows explorer	Opens an instance of windows explorer for the location of the selected script
Rename	Rename the script (also on file system)
Delete	Deletes the selected script (also on file system)

Table 8.3: *Context menu function descriptions of a script item within the toolbox.*

8.3.3 Working with folders under the scripting folder

After selecting a folder under the scripting folder "Scripts" in the toolbox and right clicking it, you get a context menu with the options as shown in Figure 8.15. The options are described in Table 8.4.

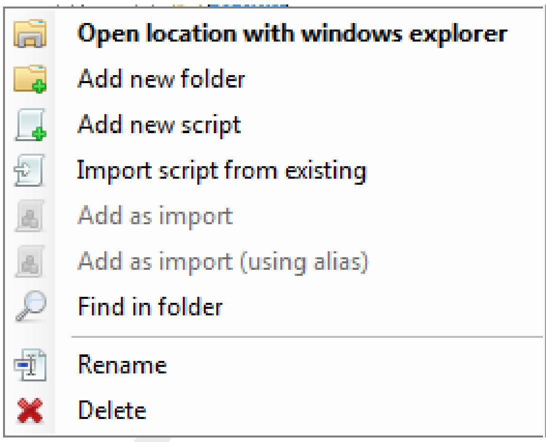


Figure 8.15: *Context menu of the "Scripts" folder (or one of its sub-folders) within the toolbox.*

Function	**Description**
Open location with windows explorer	Opens an instance of windows explorer for the location of the selected folder
Add new folder	Adds a new folder under the current selected folder and adds a __init__.py file to that folder (to make it work as import)
Add new script	Adds a new (python) script to the folder
Import script from existing	Adds a new (python) script to the folder with the content of an existing script (like a file copy)
Add as import	Adds the folder as import to the current script
Rename	Rename the folder (also on file system)
Delete	Deletes the selected folder (also on file system)

Table 8.4: *Context menu function descriptions of the "Scripts" folder (or one of its sub-folders) within the toolbox.*

8.4 Running scripts from within the Delta Shell GUI (Windows only)

After starting Delta Shell, add a script item to a new project or an existing project by right-mouse clicking on your project, and selecting *Add* → *New Item ...* shown in Figure 8.16, or add a script to your scripting folder by selecting *Add new script* from the context menu of the scripting folder within the **Toolbox** window as shown in Figure 8.15.

When adding a script to your **Project** window, a selection dialog for adding items to your project will appear. Now select the *Script* item from the various types as shown in Figure 8.17. A script item is now shown within the project. Double-click on the script item to open the script editor within the Delta Shell GUI.

Start editing the script (either in your project or in your scripting folder) using Python syntax, as shown in Figure 8.18. As shown in the figure, it is possible to start a script by importing functions from pre-defined routines developed within provided Python scripts with the Toolbox plug-in or creating your own Python libraries and adding them to your Scripts folder. These functions can than be used within in the main script. For example, in Figure 8.18, the functions `SetDefaultRoughness`, and `AddRoughnessAtLocation`, have been imported from

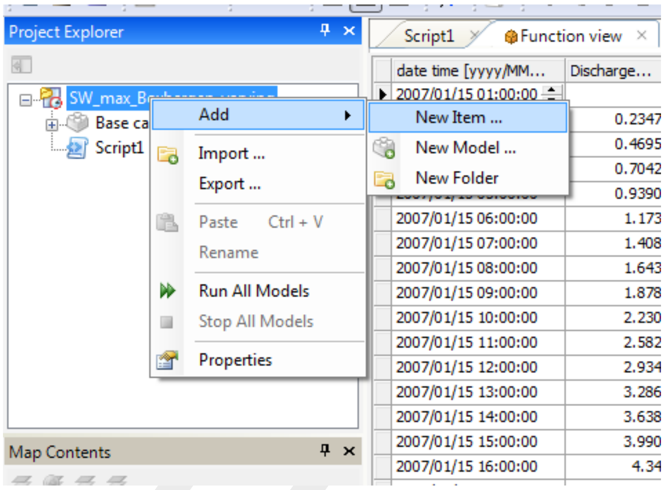


Figure 8.16: *Add an item to your Delta Shell project.*

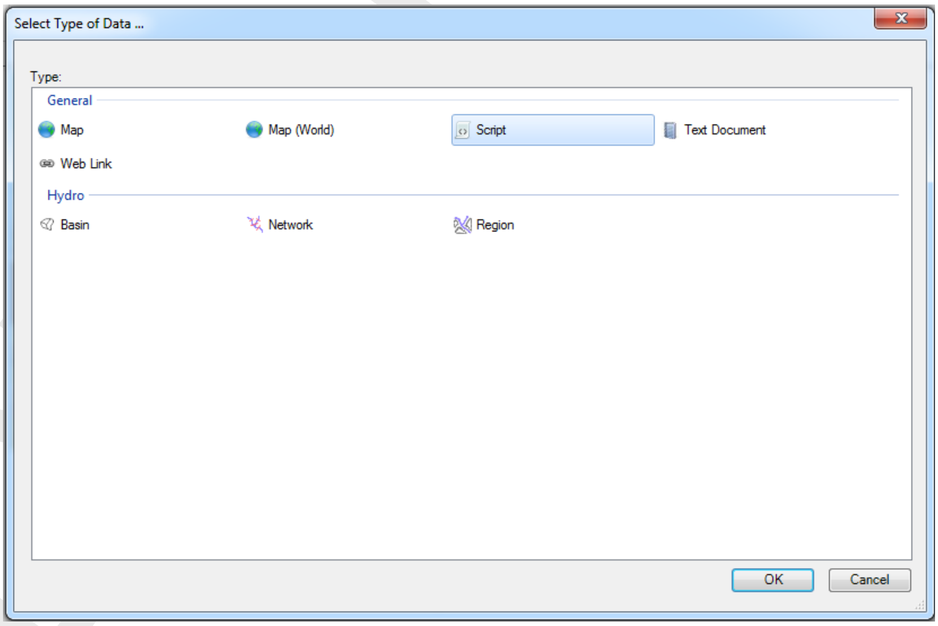


Figure 8.17: *Select script to add to your Delta Shell project.*

the pre-defined toolbox libraries as shown in the first six lines of the script.

An example of an auxiliary function as available in the pre-defined Toolbox plug-in libraries is given in Figure 8.19. The function shown needs an existing flow model and boundary name as input arguments to be able to return the desired boundary condition data to the user.

To test the script regularly during editing, or after finishing the script, the user can press the *Run script* button (within the Scripting *ribbon* in the Delta Shell GUI) shown in Figure 8.20 to see if the script is working properly. If there is something wrong within the script the messages tab within the Delta Shell GUI shows the error messages encountered during the running of the script. The messages tab will also show the messages that the user has defined himself by using `rint  \normalfont Python commands within the scrit`.

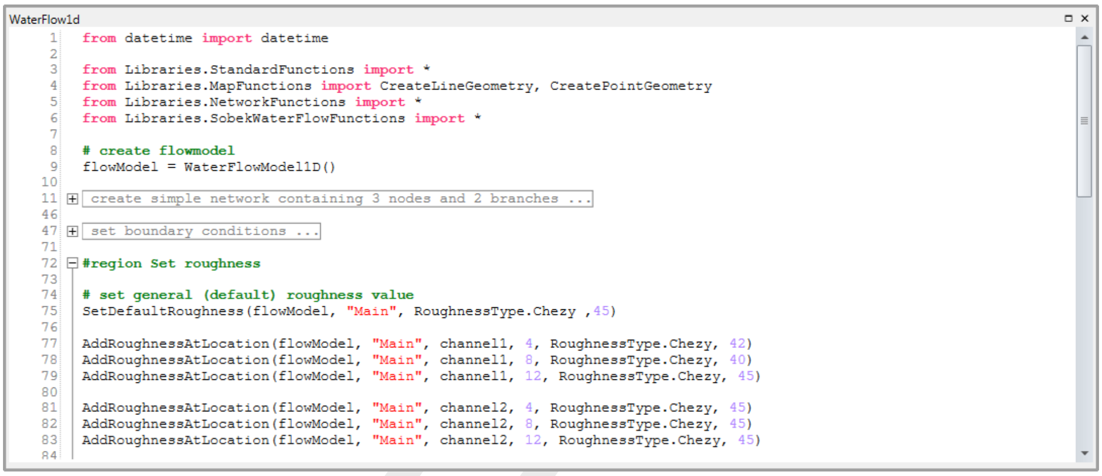


Figure 8.18: *Scripting editor within the Delta Shell GUI.*

```
def GetBoundaryDataByName(flowModel, boundaryName):
    """Gives the numeric value assigned to a Boundary Condition with the provided name"""
    for condition in flowModel.BoundaryConditions:
        if (condition.Feature.Name == boundaryName):
            boundaryData = condition
            return boundaryData
```

Figure 8.19: *Create your own auxiliary functions or use the pre-defined library functions using Python.*

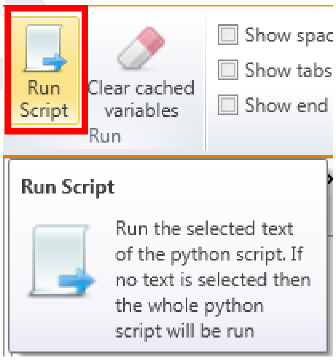


Figure 8.20: *Run script within Delta Shell GUI.*

8.5 The Delta Shell Console

Besides setting up scripts and running them from the Delta Shell GUI, users can set up scripts and run them using the Delta Shell console application. The Delta Shell console application options can be obtained from a console/terminal by typing *DeltaShell.Console.exe -?*. This will result in the options as presented in Figure 8.21.

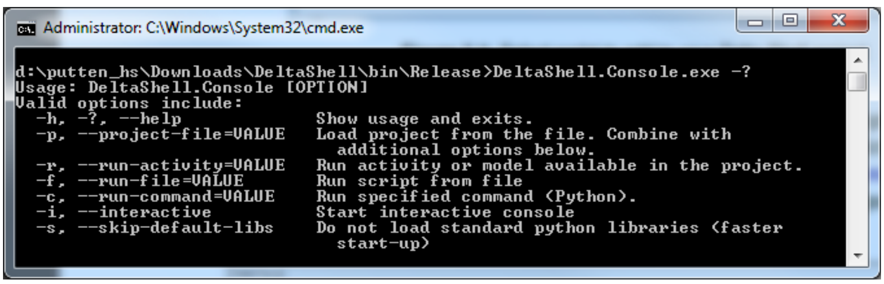


Figure 8.21: *Delta Shell console application options overview.*

8.5.1 Testing and running your scripts using the Delta Shell interactive console (Windows/Linux/MacOSX)

The Delta Shell interactive console application can be started from a console/terminal by typing *DeltaShell.Console.exe -i* within the console (make sure the Delta Shell bin directory has been added to your path or start the application within the Delta Shell bin directory of the installation) or by clicking the *Deltares* → *<Name of Software Suite>* → *Interactive Console* within the Windows start menu. Within the console application, the user can test the script command by command without losing information that has been gathered thus far, and thus can easily test these commands on a step by step basis without having to worry about making errors. The commands of the script shown in Figure 8.18 can be tested line by line in this fashion as well. After finishing the script (and saving all commands to a *.py file), run the complete script within the Delta Shell console application using the following command: *Application.ScriptRunner.RunScript((open('scriptname.py', 'r').read()))*.

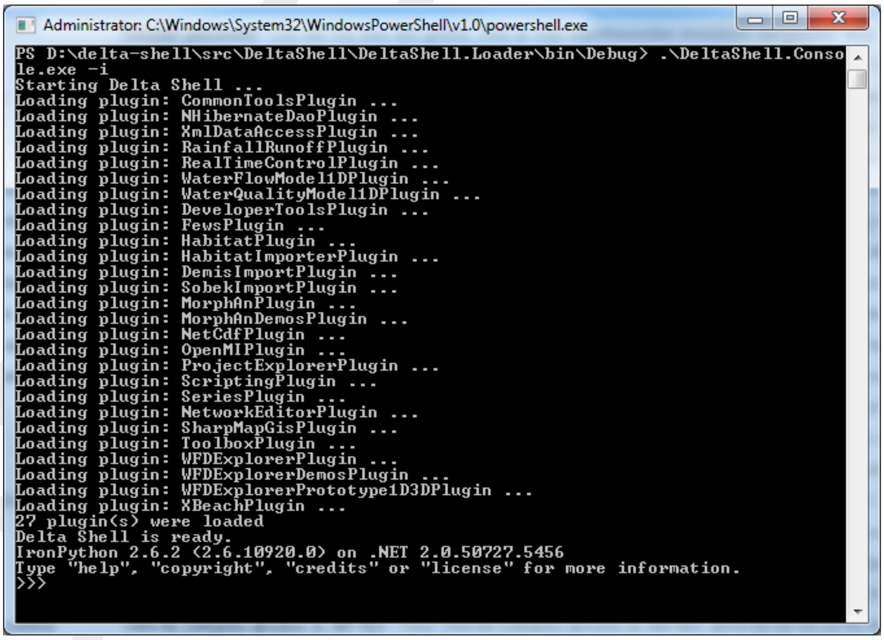


Figure 8.22: *Test script in interactive mode.*

8.5.2 Running your script directly from the commandline (Windows/Linux/MacOSX)

A finished or already existing Python script can easily be run using the Delta Shell console application as well, by passing the script to the application directly as an input argument. The command to do this from a console/terminal is: *DeltaShell.Console.exe –run-file=script.py* as is also shown in Figure 8.23.

Finally, note that when running scripts using the Delta Shell GUI, the user may also use Delta Shell GUI components within the scripts. When running scripts from the Delta Shell console application or from the command-line, these components are not available, and the user needs to fall back on GUI components (for example for plotting) available within Python or develop them himself.

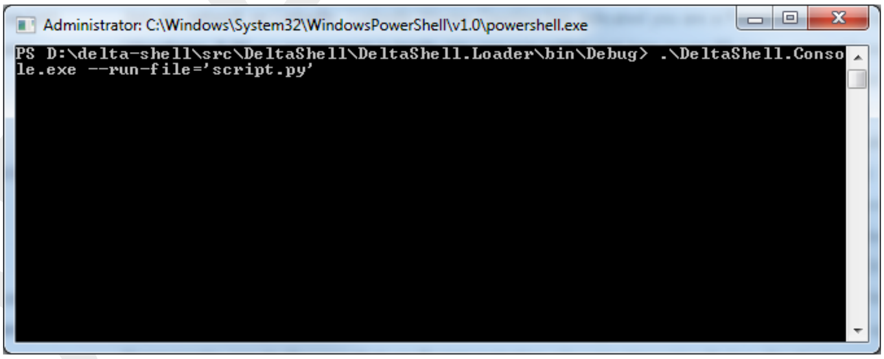


Figure 8.23: *Run script from command line within console application.*

References

Morita, M. and B. Yen, 2002. “Modeling of Conjunctive Two-Dimensional Surface-Three-Dimensional Subsurface Flows.” *Journal of Hydraulic Engineering* 128 (2): 184–200. DOI: 10.1061/(ASCE)0733-9429(2002)128:2(184).

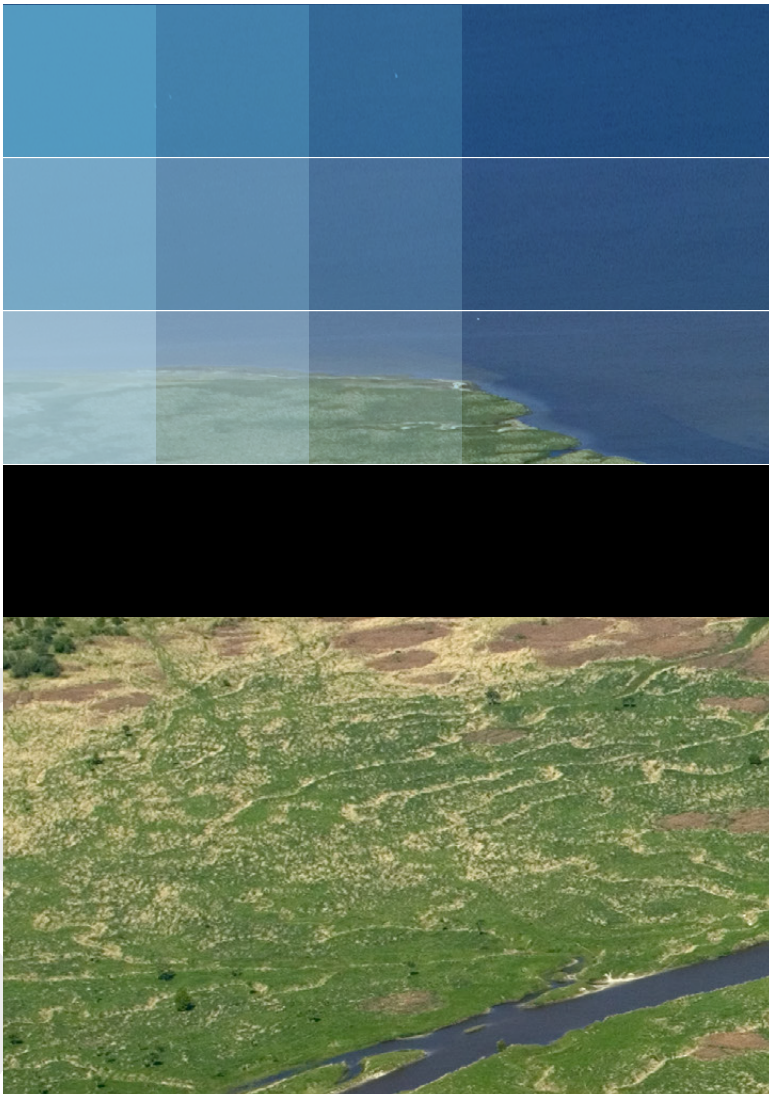
A How to use OpenDA for Delta Shell models

OpenDA is an open-source software tool distributed by the OpenDA Association (see www.openda.org). It enables the user to calibrate and Ensemble Kalman Filter (EnKF) simulation models, such as Delft3D Flexible Mesh Suite and SOBEK 3. In this document we will speak of Delta Shell (models).

Both the calibration of Delta Shell models and running them in EnKF-mode is done by using OpenDA. To run an OpenDA calibration or EnKF-simulation, a so called OpenDA application (.oda) file is needed, in which the application to be performed is specified. This oda file is the top of a hierarchy of configuration files that is organized in a directory structure that is usually setup as indicated below.

- **topDir**, containing e.g. for EnKF of a SOBEK 3 model textitSimpleModel:
  - <SimpleModelEnKF.oda>)
  - **algorithm** contains the configuration file(s) for the calibration or EnKF algorithm
  - **stochObserver** contains the configuration file(s) and measurement data for the so called ‘stochastic observer’, the set of measurements and the specification of their uncertainty
  - **stochModel** contains the configuration file(s) for the so called ‘stochastic model factory’, that specify how model instances can be created. For Delta Shell models, this is described in
    - **template** contains the base model (<SimpleModel.md1d> and the associated model files)
    - **bbWrapperConfig.xml** describes the way the model can be addressed. For Delta Shell models the so-called *black-box* wrapper is used.
    - **bbModelConfig.xml** specifies the <*.md1d> model file and some other optional settings for repeatedly running the model.
    - **bbStochModelConfig.xml** describes which items can be calibrated, and specifies the relation between the measurement series and the related observation point in the model

For the all over structure and the content of the various files, the user is referred to the documentation of OpenDA on www.openda.org. Up-to-date examples can be found there.



s systems

+31 (0)88 335 81 88
software@deltares.nl
www.deltares.nl/software