JewelSuite – Programming Challenge

Introduction

The programming test described below is a reflection of how we develop software at Baker Hughes in the Radiant project; *Get it working, get it right, optimize.* As such, there are several steps in progressing difficulty. You can stop after completing a few steps, or you can complete all of them. However, your personal time is precious, so feel free to send in what steps you completed.

The success criteria for this programming challenge are, in order of importance:

1. Calculation results (i.e. correct volume calculated in the units requested),
2. Coding style, efficiency, and quality,
3. UI, usability, look and feel.

Please write a C# or C++ application that implements the functionality in steps described below, preferably using Visual Studio 2017 or 2019 or Visual Studio Code. The user interface is preferably written in XAML. We expect the code to be of production quality.

Step 1. Enriching Data

JewelSuite has an importer for several types of data file formats for rectangular 2D grids, industry standard types and proprietary. However, at times a dataset is provided by a third party that needs extra metadata. Consider the data found in Appendix A. It lists 16 by 26 depth values in [ft] (foot). It describes the *Top horizon*, a boundary surface separating two layers with distinct rock properties.

To import the data, we need to add Inline Index, Crossline Index, Easting, Northing[1], and finally the Depth Value. Figure 1 shows the Top horizon with the depth values as a property. The *Base horizon* is formed by taking a copy of the Top horizon and lowering its depth values by 100 [m] and similarly the *Fluid Contact* is formed by setting all its depth values to 3000 [m].

[1] For more information, see the following links: Easting and Northing, Inline and Crossline.

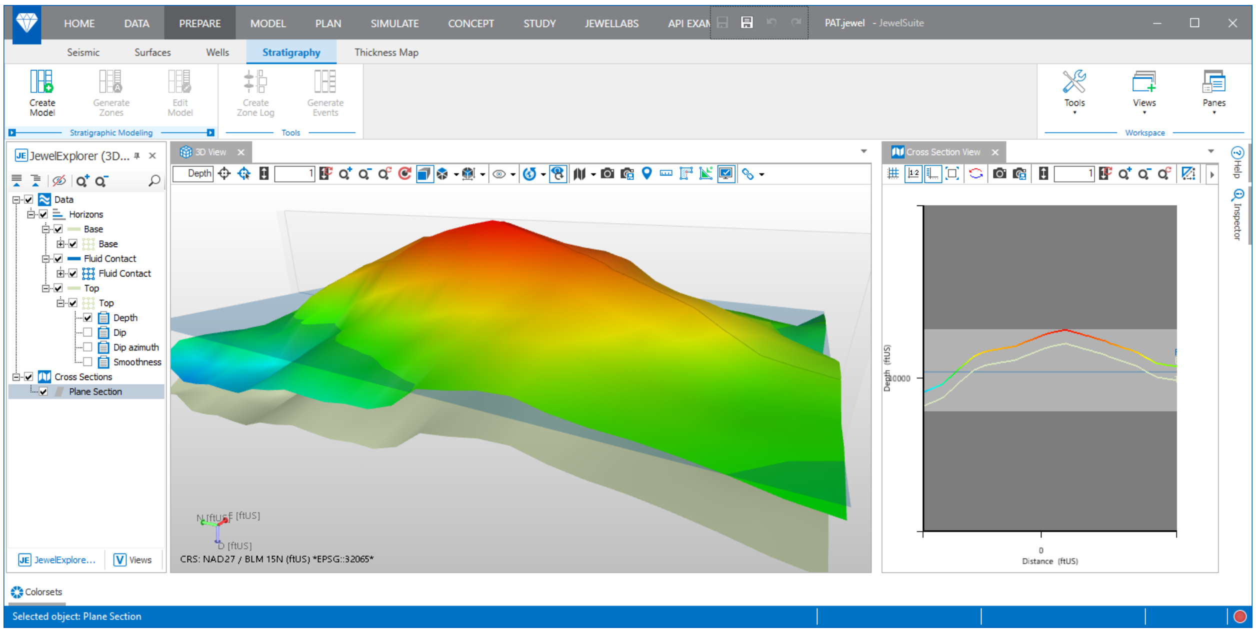
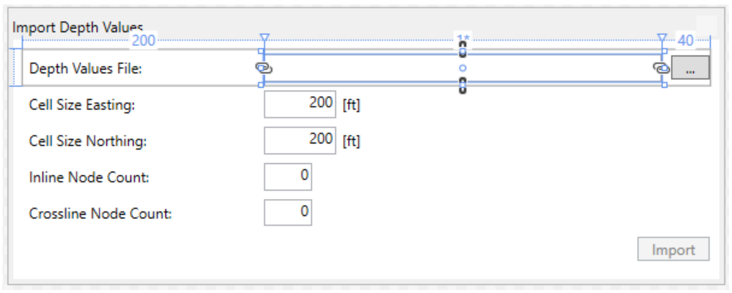


Figure 1 JewelSuite Displaying the Top Horizon, Base Horizon, and Fluid Contact

In the right pane, a cross section of the three surfaces is shown. As can be seen, there is an enclosure **between** the two horizons **above** the fluid contact. The volume of the enclosure is computed in Step 3.

Given the aforementioned data, the Inline Index runs from 0 to 25 while the Crossline Index runs from 0 to 15. Our data supplier informed us that the distance between the measurement points was 200 [ft] in Easting and also 200 [ft] in Northing. The inline and crossline directions perfectly coincided with Easting and Northing.

Write an app that imports the depth values from Appendix A. As an initial version, when pressing the Import button, the depth values are enhanced with metadata and written to a file for verification. Our Scrum teams have access to member of the UX team. The design found in Figure 2 below is their proposal. When the […] button is clicked, the user can select the file name and path.



Import Depth Values

Depth Values File:		...
Cell Size Easting:	200	[ft]
Cell Size Northing:	200	[ft]
Inline Node Count:	0	
Crossline Node Count:	0	

Import

Figure 2 Dialog Design

After the file is selected, the selected file is displayed and the number of measurement points (nodes) are updated for the inline direction and the cross line direction. The latter two values are the column count and row count, respectively. The user can specify the cell sizes of the 2D grid, i.e., the distance between the measurement points in Easting and in Northing, which align in this case with inline and crossline, respectively. The unit [ft] is hard coded, for simplicity. After pressing the [Import] button, a dialog is shown to select the file name and path where to store the file with the enriched data. If the user pressed OK, the enriched data file is created and stored at the given file path. You can use ReactiveUI if you like to.

Please implement this design and functionality for Step 1.

Step 2. Additional Surfaces

The screenshot of JewelSuite shows the Top horizon after successfully importing the data file produced in the previous steps. We copied the Top horizon to create the Base horizon and lowered it by 100 meter using the Property Calculator. Next we copied the Top horizon once more to create the Fluid Contact and set each location to a depth of 3000 meter. Finally, we assigned some colors.

In your app, modify the existing dialog, or add another dialog, to create the Base horizon and the Fluid Contact. The user should be able to set distance between the Top horizon and the Base horizon as well as the (uniform) depth of the Fluid Contact.

Step 3. Volume

Assume oil is present in the enclosure of rock above the Fluid Contact between the Top horizon and the Base horizon. Compute this volume. Display the volume specified in [m^3], [ft^3], and US barrel.

Step 4. Quality Assurance

Writing production code means quality assurance and therefore in TTD way of working, we write unit tests. One test is to make sure that the correct volume is computed in Step 3.

Create a data file with two lines with two times the value 0. Import this file in your app. Set both step sizes to 1 ft. Next, set the distance between the Top horizon and the Base horizon to 1 [ft]. Finally, set the depth of the Fluid Contact to 1 [ft]. This creates a cube of exactly 1 [ft^3]. If all went well in Step 1 through Step 3, you should see that answer.

Step 5. Refactor

With QA tests in place, we can start to refactor. The code behind found for the [Import] button did three things:

- Select a file name and path,
- Create the enriched data,

- Write the enriched data to the given file.

That is not very SOLID and we want as little code behind as possible. So, let's refactor it.

To start, add ImportFilePath as a member of your view model. Create a service with one method that takes an enumeration of strings with the depth values, and produces a domain object called Grid2D. Additional parameters for the service method are EastingStepSize and NorthingStepSize. Grid2D has InlineCount and CrosslineCount as properties and a property that for a given inline index and crossline index returns a 3D location with the values Easting, Northing, and Depth. Please keep the internal data in SI units, meters in this case.

Create a service that produces a Grid2D from a series of depth values.

Finally, write another service that takes a Grid2D instances, and writes each of its locations (vertices) to a text file. If you want you can convert the location values into to foot or export the SI values directly.

Step 6. Layering

At Baker Hughes, we write plugins for JewelSuite using our so-called *Reference Architecture*. That means the code is separated into three layers; UI code goes in the Presentation layer, commands and query services go in the Application layer, and domain objects into the Domain layer.

The Reference Architecture splits also splits each layers three; API, Core, and Logic, each with a separate project. Dependency Injection is used to connect interfaces to implementing classes. We also use CQRS to modify domain objects. That involves splitting the command and the command handler. Assuming a CQRS framework is not available for this programming test, and that writing that from scratch is too much work, we simplify things in the current step. However, if you want, go for it.

Split the code you produced in Step 1 through Step 3 into these layers by creating three projects and placing the code at the right layer. If you want, you can go for Dependency Injection / Inversion of Control and add additional API projects.

Appendix A

The following list of 16 by 26 numbers are the depth values data in foot. It represents a two dimensional (2D), regular grid with the depth value at the node. The 2D grid cell size is 200 by 200 [ft]. The origin can be set at (0, 0) in Easting, Northing.

```
9911 9867 9824 9818 9767 9704 9691 9754 9843 10055 10082 10006 9865 9833 9842 9871
9889 9845 9804 9796 9746 9677 9644 9672 9753 9903 9998 9961 9854 10050 10039 10053
9845 9807 9792 9784 9732 9641 9580 9590 9639 9658 9757 9808 9824 10186 10421 10399
9811 9765 9747 9720 9625 9518 9498 9512 9570 9641 9679 9757 9782 10143 10464 10452
9786 9735 9698 9660 9549 9377 9381 9422 9491 9609 9677 9724 9761 10124 10417 10428
9776 9712 9656 9591 9472 9293 9320 9370 9437 9508 9588 9652 9734 10129 10409 10388
9749 9682 9606 9526 9401 9209 9227 9273 9367 9434 9507 9580 9688 10090 10412 10364
9698 9619 9532 9425 9305 9135 9100 9123 9188 9358 9451 9542 9645 10035 10354 10354
9714 9618 9525 9415 9211 9059 9047 9078 9115 9279 9409 9509 9604 9996 10316 10336
9765 9665 9570 9457 9245 8974 8996 9049 9135 9272 9395 9494 9563 9957 10281 10322
9805 9694 9594 9472 9207 8896 8926 9026 9119 9228 9347 9451 9529 9951 10270 10306
9982 9734 9616 9489 9217 8826 8860 8964 9106 9189 9290 9407 9499 9916 10268 10291
10066 9817 9697 9543 9266 8869 8904 8985 9113 9182 9262 9419 9656 9841 10176 10214
10078 9883 9758 9607 9327 8924 8965 9022 9118 9175 9238 9380 9702 9929 10085 10137
10118 9933 9778 9638 9373 9000 9062 9088 9140 9149 9194 9319 9653 9868 10017 10069
10120 9979 9811 9685 9460 9104 9144 9150 9153 9127 9171 9281 9616 9809 9958 10013
10115 9987 9853 9743 9532 9234 9245 9229 9186 9091 9130 9262 9596 9771 9893 9966
10172 10067 9949 9863 9624 9323 9283 9237 9185 9103 9126 9274 9627 9786 9898 9972
10211 10146 10060 9969 9759 9404 9333 9256 9178 9109 9139 9288 9667 9827 9921 9992
10302 10259 10200 9803 9717 9463 9389 9331 9268 9243 9263 9413 9804 9866 9935 9980
10313 10264 10235 9916 9654 9560 9493 9435 9396 9390 9410 9530 9927 9997 9972 9993
10316 10216 10035 10013 9812 9736 9701 9647 9611 9627 9615 9708 10032 10110 10082 10046
10360 10239 9986 9956 9964 9904 9900 9904 9883 9928 9842 9876 10157 10200 10179 10122
10383 10278 10040 10014 10029 10057 10099 10134 10179 10285 10168 10095 10271 10295 10255 10202
10252 10251 10076 10079 10105 10156 10219 10236 10293 10399 10315 10205 10367 10363 10320 10270
10287 10094 10097 10123 10170 10235 10312 10342 10351 10457 10377 10292 10436 10424 10374 10330
```