**Functionality Outline**

Oil Well Simulation

**Program Assignment** 2

**Prepared By**

Jaiden Gann

[jmg0053@uah.edu](mailto:Jtw0014@uah.edu)

**Prepared for**  
**Mr. James Williamson**  
**CS 307, Object Oriented Programming**  
**Computer Science Department**  
**University of Alabama in Huntsville**

**1.0 System Overview……………………………………………………………………………………..……….3**

**2.0 Relevant Terms and Acronyms…………………………………………………………………..……..3**

**3.0 Object Functionality………….…………………………………………………………………..………….4**

**3.1 StartSim…………………..………………………………………….……………………..…………4**

**3.2 RunProgram…………………………………………………..…………………………….…....…4**

**3.3 Display………………………………………………………………………………….………….…..5**

**3.4 Report………………….………………………………………………………………………….......5**

**3.5 SensorFactory………………………………………………………………………….……..…....5**

**3.6 Sensor……………………………………………….………………….……………………...…...…6**

**3.7 WellFactory…………………………………………………………………………….…….….……7**

**3.8 Well…………………………………………………………………………………………………….…7**

**3.9 DataGenFactory……………………………………………………………………………..……..8**

**3.10 FollowLink……………………………………………………………………………………………8**

**3.11 Random………………………………………………………………………………………………..8**

**3.12 StepInc………………………………………………………………………………………………….9**

**3.13 StepDec………………………………………………………………………………………………..9**

## 1.0 System Overview

The purpose of the program is to simulate software used to monitor sensor readings on a number oil rigs. The sensors constantly output data to a display for works and managers to monitor the aspects of rig operations. This program will be developed with as much object-oriented functionality as possible so that it can be expanded and changed easily for the future.

## 2.0 Relevant Terms and Acronyms

*Rig* – Refers to a single oil well. Can be used interchangeably with “well”

*ID* – Refers to a number sequence used to Identify each well. Will succeed the name of the owning company

*Hole Depth* – Depth of the hole that has been drilled. Measured in feet (ft)

*Bit Depth* – Current depth(location) of the drill head. Measured in feet (ft)

*ROP* – rate of penetration. Used to indicate how fast the drill head is drilling through the crust. Measured in feet per hour (ft/hr)

*Off Bottom* – indicates the drill bit is not currently drilling and ROP is 0 ft/hr

*Pump Pressure* – Pressure in the lubricant(mud) pump. Measured in pounds per square inch (PSI)

*Casing Pressure* – Pressure in the drill bit casing itself. Measured in pounds per square inch (PSI)

*Flow out* – Mud(lubricant) flowing out of the bit casing. Measured in percentage of maximum flow (%)

*Torque Max* – Maximum torque that can be safely applied to the drill bit. Measured in kilo-foot pounds (kft-lbs)

*Mud Pit Volume* – Volume of mud(lubricant) available. Measured in barrels (BBL)

## 3.0 Object Functionality

### StartSim.cpp

**Main()**

Create instance of simulation to run

Run the run function from runprogram

### RunProgram.cpp

**Simulation()**

Create the simulation object

**run(filename)**

read the input file

get an instance of the WellFactory

**update()**

update each well in the vector of wells

**vector <Well \*> getWells()**

return wells

**editWells()**

char \*add\_rem = new char[32]

char \*id = new char[32]

bool enabled = false

output current Id list

prompt for add or remove

prompt for well Id

enable or disable based on input and return error for wrong input

enable well in the vector of wells

**editSensor()**

same as wells except do it for sensors

given well Id (if statement)

output sensor types on it

enable sensor on well

**reportLogs()**

if(wellEnabled)

new WellReport

this.display.Info(wellReport)

### Display.cpp

**Display(ostream \*out)**

\_out = out

**Info(Report \*data)**

Output wellInfo and sensor Info from report.cpp

### Report.cpp

**wellReport(\*well)**

create new \_well\_info character array

print \_well\_info //getCompany, and geID functions used from well

for each sensor

print the enabled sensors using a vector (\_sensor\_info) //sensor functions getDisplayName, getValue, get Abbrev used

**~WellReport()**

Delete \_well\_info

Clear the sensor vector used to print sensor info

**getWellInfo()**

return \_well\_info

**getSensorInfo()**

return \_sensor\_info

### SensorFactory.cpp

**~SensorFactory()**

**Static getInstance()**

create instance = nullptr

if(instance == nullptr)

instance = new SensorFactory()

return instance

**createSensor(char \*type)**

create new Sensor of given type

**SensorFactory()**

### Sensor.cpp

**Sensor(sensor\_data)**

Create char arrays for variables: \_value, \_type, \_abbrev, \_units, \_sensorName

Copy the input into the variables

Set min and max to their variables

Get an instance of the DataGenFactory

From the instance of datagenfactory createDataGen(alg, init value, max, min)

**Update()**

Generate new data

**~Sensor()**

**getName()**

return \_sensorName

**getValue()**

return \_value

**IsEnabled()**

Return \_enabled

**getType()**

return \_type

**getUnitsAbbrev()**

return \_abbrev

**getUnits()**

return \_units

### WellFactory.cpp

**~WellFactory()**

**Static getInstance()**

create instance = nullptr

if(instance == nullptr)

instance = new WellFactory()

return instance

**createWell(\*data)**

create necessary well and sensor variables

data.getWellData(id, opr, &num\_sensors, types)

for each number of sensors

data.getSensorData(//comes from data parser)

well.addSensor(\_sensor\_factory.createSensor(\*types[i])

**WellFactory()**

Associate \_sensor\_factory with instance an instance of SensorFactory

### Well.cpp

**Well(char \*id, char \*company, int num\_sensors)**

Set up new char arrays for \_id and \_company

Copy the \_id and \_company to id and company

Set \_sensors\_count = num\_sensors

**getCompany()**

return \_company

**getSensorscount()**

return \_sensors\_count

**getId()**

return \_id

**IsEnabled()**

Return \_enabled

**setEnable(bool e)**

\_enabled = e

**setSensorEnabled(char \*name, bool e)**

for the sensors

if the given name matches a sensor name

enable it

**update()**

call each sensor’s update function

**getSensorTypes()**

create a vector to hold results

for each sensor

add sensor type to vector

return the vector

### DataGenFactory.cpp

**DataGenFactory()**

**~DataGenFactory()**

**getInstance()**

create instance = nullptr

if(instance == nullptr)

instance = new DataGenFactory()

return instance

**createData(alg, int\_value, max, min)**

if alg matches one of the algorithm names return new of that algorithm function

return null if there isn’t a match

### FollowLink.cpp

**Generate()**

Return (alg) //algorithm should be provided to us

### Random.cpp

**Generate()**

Return (alg) //algorithm should be provided to us

### StepInc.cpp

**Generate()**

Return (alg) //algorithm should be provided to us

StepDec.cpp

**Generate()**

Return (alg) //algorithm should be provided to us