**Glass Size Estimator**

User Guide

# 1. Introduction

The Glass Size Estimator is a WinForms application designed to run on Windows machines. This application eliminates the human error from estimating glass door cuts by calculating the measurements automatically based on the inputs provided by the user. This application was designed to read the data of JSON configuration files containing product lines containing a list of expected inputs, expected outputs, and a state machine defining the logic to calculate the outputs given the values provided for the inputs by the user. All product lines in the configuration file are able to be selected from a list by the user. The inputs and outputs are added to the application form dynamically when a product line is clicked based on the input/output properties defined in the configuration file. This design of making the application driven by the data of the configuration files allows the client to edit these configuration files as more product lines are required for users, with no code changes necessary.

# 2. Installation Guide

1. Run ‘GlassEstimatorSetup.msi’
2. Navigate to the installation location
   1. ‘C:/Program Files (x86)/HMICardinal/GlassEstimatorSetup’ or ‘C:/Program Files/HMICardinal/GlassEstimatorSetup’
3. Copy ‘product\_line\_config.json’ to the directory
   1. This is the file that can be modified to adjust and add product lines
4. Copy ‘stock\_glass\_line\_config.json’ to the directory
   1. This is the file that can be modified to adjust and add stock information
5. Run ‘Glass Size Estimator.exe’

# 3. Application Details

The Glass Size Estimator requires a computer running any version of the Windows operating system that has the .Net Framework version 4.6 installed.

## 3.1. Application Workflow Diagram

# https://lh6.googleusercontent.com/qbaoXnaiKkMy1LCFyhbklSQWAppa2yQeUme4gRbrcM4G6A801xHjl7YuTqGmB9lxEuYhPseulr2OUvc3koBsGmAFKEziilOA7GURBwaBobN4Y2CCMJnoS3HPiaHOB7073R5VUVEx

# 4. Product Lines

## 4.1. Introduction

All product lines that populate the selection list in the application are derived from JSON files (e.g. product\_line\_config.json) located in the same directory as the application. As a result, changes to the product lines can be made via a text editor or online JSON editor with little to no changes to the code.

## 4.2. Configuration Fields

The sections below describe all of the fields required in the configuration to implement a product line in the application.

Each product lines should be defined in the array of ProductLines.

### {

### "ProductLines": [

### {

// Define each product at this level in the hierarchy

}

]

}

### 4.2.1. Name

This field represents the name of the product line.

#### 4.2.1.1 Name Example

"Name": "Stock Cardinal Series Semi-frameless Single Doors"

### 4.2.2. Input

This field describes the types of input used by the corresponding product line.

Each input requires a **name** and **type** field. An additional **options** field is required for enumerations.Any name can be provided in the configuration. The types currently supported for inputs include **integer** (whole number), **float** (decimal numbers), **boolean** (true or false values), and **enum** (list of options).

To work, options provided to enum inputs must match with the enumerations described in the application. Currently, the defined enumeration categories for the application include **Configuration**, **GlassType**, **Series**, and **WallJamb**. Therefore, changes to the code are required to add new enumerations. A new enum has to be implemented and mapped in EnumManager.cs.

#### 4.2.2.1 Input Example

"Input": [

{

"Name": "OpeningWidth",

"Type": "Float"

},

{

"Name": "OpeningHeight",

"Type": "Float"

},

{

"Name": "ClearSweep",

"Type": "Boolean"

},

{

"Name": "TwoHoles",

"Type": "Boolean"

}

]

### 4.2.3. Output

This field describes the types of output used by the corresponding product line.

Each input requires a **name**, **type**, and **input** field. Any name can be provided in the configuration. The types currently supported for outputs include **integer** (whole number), **float** (decimal numbers), **boolean** (true or false values), and **enum** (list of options). The input is the value placed into the logic pipeline. Therefore, it must match the name of an input provided in the section above.

#### 4.2.3.1 Output Example

"Output": [

{

"Name": "ResultingWidth",

"Type": "Float",

"Input": "OpeningWidth"

},

{

"Name": "ResultingHeight",

"Input": "OpeningHeight",

"Type": "Float"

},

{

"Name": "WallJamb",

"Input": "OpeningWidth",

"Type": "Enum"

}

]

### 4.2.4. Logic

This field describes the logic that is followed by each output. The logic for each product line is described as a state machine. The logic flows similar to code execution in Assembly or a choose-your-own-adventure book (i.e. if X is true, go to state 5; else, go to state 6).

Each state machine is an array with the same name as an output described in the previous section. Afterwards, each object within the array describes one of many states, which perform one specialized action (e.g. Adding a number, Rounding up a number, and Moving to a new state). Attempting to use states that do not exist, trying to jump to a nonexistent position in the state machine, and providing the wrong data type to a state will result in unexpected errors in the application. It is recommended to double check the configuration before attempting to execute the defined logic in the application.

### 4.2.5. States

All states require the **Operation** field.

**Operation** represents the name of the state that will be placed in the state machine.

Moreover, the information provided by the user is separated into two categories, Pipeline and Parameters. The value of **Input** specified for the given output will be placed in the pipeline, will be modified, and returned under the name specified. Other values, such as checkboxes and dropdown selections, are placed in parameters. These values can be used by other states on the side to make decisions without affecting the value that will be returned as output.

The following subsections describe each state implemented in the application and any additional required fields.

#### 4.2.5.1. Addition

Add the given number to the current value in the pipeline.

Requires a **Value** field.

**Value** represents the number that will be used in the addition operation.

#### 4.2.5.2. BranchConditional

Look at the specified conditional and branch accordingly.

Requires a **NextState**, **Qualifier**, **ConditionalName** field.

**NextState** represents the next state that the state machine will go to once the current state is finished processing and if the desired qualifier is met.

**Qualifier** represents the desired value (true or false) of the targeted conditional.

**ConditionalName** represents the name of the conditional that will be compared to qualifier.

#### 4.2.5.3. BranchEnum

Look at the specified enum in the parameters and branch accordingly.

Requires a **NextState**, **Qualifier**, **EnumCategory**, and **EnumList** field.

**NextState** represents the next state that the state machine will go to once the current state is finished processing and if the desired qualifier is met.

**Qualifier** represents the desired result (true or false) from checking whether or not an enum in the given parameters is contained within the given list of options in the given enum category has been provided by the user.

**EnumCategory** represents the category of enums that will be checked (i.e. WallJamb, Series, and Configuration).

**EnumList** represents an array of options that will be used in the comparison.

#### 4.2.5.4. BranchFractional Value

Look at the specified fractional portion of the value in the pipeline and branch accordingly.

Requires a **NextState**, **Qualifier**, **Minimum**, and **Maximum** field.

**NextState** represents the next state that the state machine will go to once the current state is finished processing and if the desired qualifier is met.

**Qualifier** represents the desired result (true or false) from checking whether or not the fractional portion of the value in the pipeline is in between the given minimum and maximum values (inclusive).

**Minimum** represents the lower end of the range that will be checked.

**Maximum** represents the upper end of the range that will be checked.

#### 4.2.5.5. BranchInputValue

Look at the specified value in the given parameters and branch accordingly.

Requires a **NextState**, **Qualifier**, **Minimum**, **Maximum**, and **InputName** field.

**NextState** represents the next state that the state machine will go to once the current state is finished processing and if the desired qualifier is met.

**Qualifier** represents the desired result (true or false) from checking whether or not the the value in the given parameters is in between the given minimum and maximum values (inclusive).

**Minimum** represents the lower end of the range that will be checked.

**Maximum** represents the upper end of the range that will be checked.

**InputName** represents the name of the desired parameter that will be examined.

#### 4.2.5.6. Branch

Branch to another state immediately.

Requires a **NextState** field.

**NextState** represents the next state that the state machine will go to once the current state is finished processing.

#### 4.2.5.7. BranchValue

Look at the value in the pipeline and branch accordingly.

Requires a **NextState**, **Qualifier**, **Minimum**, and **Maximum** field.

**NextState** represents the next state that the state machine will go to once the current state is finished processing and if the desired qualifier is met.

**Qualifier** represents the desired result (true or false) from checking whether or not the value in the pipeline is in between the given minimum and maximum values (inclusive).

**Minimum** represents the lower end of the range that will be checked.

**Maximum** represents the upper end of the range that will be checked.

#### 4.2.5.8. Division

Divide the current value in the pipeline by a given number.

Requires a **Value** field.

**Value** represents the number that will be used in the division operation.

#### 4.2.5.9. End

Signals the end of the state machine. Once the end is reach, the current value in the pipeline will be returned to the user in the corresponding output field.

Requires no additional fields.

#### 4.2.5.10. Multiplication

Multiply the given number to the current value in the pipeline.

Requires a **Value** field.

**Value** represents the number that will be used in the multiplication operation.

#### 4.2.5.11. RoundDown

Round down the current value in the pipeline to the given interval.

Requires an **Interval** field.

**Interval** represents the interval that the number will be rounded down to (i.e. if the given interval is .5, 8.7 will be rounded down to 8.5; if the given interval is .125, 8.7 will be rounded down to 8.625)

#### 4.2.5.12. RoundUp

Round up the current value in the pipeline to the given interval.

Requires an **Interval** field.

**Interval** represents the interval that the number will be rounded down to (i.e. if the given interval is .5, 8.8 will be rounded up to 9.0; if the given interval is .125, 8.8 will be rounded up to 8.875)

#### 4.2.5.13. SetConditional

Set the current value in the pipeline to the given conditional value.

Requires a **Value** field.

**Value** represents the value (true or false) that will replace the current value in the pipeline.

#### 4.2.5.14. SetEnum

Set the current value in the pipeline to the given enum.

Set the current value in the pipeline to the given conditional value.

Requires a **Value** and **Category** field.

**Value** represents the enum that will replace the current value in the pipeline.

**Category** represents the category the given enum belongs to (i.e. WallJamb, Series, and Configuration).

#### 4.2.5.15. SetValue

Set the current value in the pipeline to the given number.

Set the current value in the pipeline to the given conditional value.

Requires a **Value** field.

**Value** represents the value that will replace the current value in the pipeline.

#### 4.2.5.16. Subtraction

Subtract the given number to the current value in the pipeline.

Requires a **Value** field.

**Value** represents the number that will be used in the subtraction operation.

#### 4.2.5.17. Truncate

Truncate the current value in the pipeline.

Requires no additional fields.

#### 4.2.5.18 Logic / State Example

"Logic": {

"ResultingWidth": [

{

"Operation": "BranchInputValue",

"Minimum": 66.625,

"Maximum": 66.625,

"NextState": 4,

"Qualifier": true,

"InputName": "OpeningHeight"

},

{

"Operation": "BranchInputValue",

"Minimum": 69.625,

"Maximum": 69.625,

"NextState": 4,

"Qualifier": true,

"InputName": "OpeningHeight"

},

{

"Operation": "BranchInputValue",

"Minimum": 72.625,

"Maximum": 72.625,

"NextState": 4,

"Qualifier": true,

"InputName": "OpeningHeight"

},

{

"Operation": "Branch",

"NextState": 16

},

{

"Operation": "BranchFractionalValue",

"Minimum": 0,

"Maximum": 0.125,

"NextState": 7,

"Qualifier": true

},

{

"Operation": "BranchFractionalValue",

"NextState": 9,

"Minimum": 0.1875,

"Maximum": 0.6875,

"Qualifier": true

},

{

"Operation": "BranchFractionalValue",

"NextState": 11,

"Minimum": 0.75,

"Maximum": 1,

"Qualifier": true

},

{

"Operation": "RoundDown",

"Interval": 1

},

{

"Operation": "Branch",

"NextState": 13

},

{

"Operation": "Truncate"

},

{

"Operation": "Branch",

"NextState": 13

},

{

"Operation": "RoundUp",

"Interval": 1

},

{

"Operation": "Branch",

"NextState": 13

},

{

"Operation": "Subtraction",

"Value": 4

},

{

"Operation": "Addition",

"Value": 0.8125

},

{

"Operation": "Branch",

"NextState": 17

},

{

"Operation": "Subtraction",

"Value": 3.5

},

{

"Operation": "End"

}

]

}

### 4.2.6. Category

This field describes the category of stock glass that the results from the product line will be compared to (e.g. Door or Panel).

#### 4.2.6.1 Category Example

"Category": "Door"

# 5. Stock Glass Lines

## 5.1. Introduction

Similar to the product lines, all stock glass lines information that is utilized in the application are derived from JSON files (e.g. stock\_glass\_line\_config.json) located in the same directory as the application. As a result, changes to the stock glass lines can be made via a text editor or online JSON editor with little to no changes to the code.

This resulting heights and widths from the operation are compared against the measurements defined in this configuration file.

## 5.2. Configuration Fields

Each stock glass line is represented by an array named after the stock glass line. Afterwards, inside are various objects with only two fields **Width** and **Height** that are used to represent each available dimension in stock.

### 5.2.1. Configuration Example

{

"Door\_Glass\_69\_Stall\_3/16\_Clear": [

{

"Width": 18.8125,

"Height": 65

},

{

"Width": 19.8125,

"Height": 65

},

{

"Width": 20.8125,

"Height": 65

},

{

"Width": 21.8125,

"Height": 65

},

{

"Width": 22.8125,

"Height": 65

},

{

"Width": 23.8125,

"Height": 65

},

{

"Width": 24.8125,

"Height": 65

},

{

"Width": 25.8125,

"Height": 65

},

{

"Width": 26.8125,

"Height": 65

},

{

"Width": 27.8125,

"Height": 65

},

{

"Width": 28.8125,

"Height": 65

},

{

"Width": 29.8125,

"Height": 65

},

{

"Width": 30.8125,

"Height": 65

},

{

"Width": 31.8125,

"Height": 65

},

{

"Width": 32.8125,

"Height": 65

}

]

}

# 6. Helpful Links

* <https://jsoneditoronline.org/> - Online JSON Editor