Math And Logic in Aerofly FS 2

Aerofly FS 2 TMD programming is touring complete, which means you can create pretty much any arbitrary complexity and actually code real programs within the tmd.

There is a wide range of objects available for you to set up your digital systems or simulate new physical system within the tmd.

From simple scaling to complex logic circuits and even flight controller programming, the set of available objects has you covered.

Together with the powerful event programming even complex menu systems, iterations, integrations, delays and a lot more are possible to create.

> All of these components introduced here have an `Output` function. This makes it very easy to set up and chain together. Few of them have more than one output.

Simple Algebra

Let us introduce the simple mathematical building blocks one at a time.

constant

A value that cannot change.

```
            <[constant][StaticTrue][]
                <[string8][Input][1.0]>
            >
```

linear

Linear scale and constant offset.

```
return Input() * Scaling + Offset;
```

```
            <[linear][DoubleA][]
                <[string8][Input][A.Output]>
                <[float64][Scaling][1.0]>
                <[string8][Offset][0.0]>
            >
```

mixlinear

Sum of two values, scaled with a factor. Weights can be negative, to subtract the inputs.

```
return Input0() * Weight0 + Input1() * Weight1 + Offset;
```

```
            <[mixlinear][ABMix][]
                <[string8][Input0][A.Output]>
                <[string8][Input1][B.Output]>
                <[float64][Weight0][1.0]>
                <[float64][Weight1][1.0]>
            >
```

maximum

Returns the highest value in the list.

```
            <[maximum][MaximumOfValue0Value1][]
                <[string8][Inputs][ Value0.Output Value1.Output ]>
            >
```

minimum

Returns the lowest value in the list.

```
            <[minimum][MinimumOfValue0Value1][]
                <[string8][Inputs][ Value0.Output Value1.Output ]>
            >
```

absolute

Returns the absolute value of the input, without the sign. Inputs below 0.0 are turned positive.

```
            <[absolute][Absolute][]
                <[string8][Input][Input.Output]>
            >
```

sum

Total sum of all inputs.

```
            <[sum][APlusB][]
                <[string8][Inputs][ A.Output B.Output ]>
            >
```

product

Multiplies all inputs. Works great to turn off a function under certain conditions, e.g. disable the nose

wheel steering.

```
            <[product][ATimesB][]
                <[string8][Inputs][ A.Output B.Output ]>
            >
```

inverse

Divides 1.0 with the input. When Input is close to zero the inverse returns 0.0. Normally this function is never needed in the physics computations. If you want to convert units, this can be done in the graphics_input with the Scaling.

```
            <[inverse][OneOverA][]
                <[string8][Input][A.Output]>
            >
```

linear_interpolation

Linear interpolations or mappings work by finding an intermediate value from nearby points. The points in the map define discrete points where the output value is know. The map is a list of 2d-vectors. The first component is the input, the second is the output mapped to that input.

The linear_interpolation stops at the last value and does not extrapolate.

```
            <[linear_interpolation][LinearMapping][]
                <[string8][Input][Control.Output]>
                <[tmvector2d][Map][ (0.0 0.0) (1.0 1.0) ]>
            >
```

polynomial

Simulates a polynomial function `a_n * x^n + … + a_2 * x^2 + a_1 * x + a_0`

```
            <[polynomial][Constant][]
                <[string8][Input][Input.Output]>
                <[float64array][Coefficients][ 1.0 ]>
            >
            <[polynomial][InputDoubled][]
                <[string8][Input][Input.Output]>
                <[float64array][Coefficients][ 2.0 0.0 ]>
            >
            <[polynomial][InputSquared][]
                <[string8][Input][Input.Output]>
                <[float64array][Coefficients][ 1.0 0.0 0.0 ]>
            >
            <[polynomial][InputCubed][]
                <[string8][Input][Input.Output]>
```

```
            <[float64array][Coefficients][ 1.0 0.0 0.0 0.0 ]>
        >
```

clamp

When the value is within a certain range the value itself is returned. When it leaves the range the output stays bounded to the range.

```
        <[clamp][Clamped][]
            <[string8][Input][Value.Output]>
            <[tmvector2d][Range][ -1.0 1.0 ]>
        >
```

clamp_cyclic

When the value is within a certain range the value itself is returned. When it leaves the range the output loops back to the start. A good example is the heading on a compass rose that rotates 360° and then loops back to 0.

```
        <[clamp_cyclic][HeadingDeviation][]
            <[string8][Input][HeadingDeviationSum.Output]>
            <[tmvector2d][Range][ -3.14159 3.14159 ]>
        >
```

Dynamic Systems

Moving on from the basic functions, let's get into the more advanced mathematics.

integral

Integrates the input with time. When the input is 1.0 then the output changes by 1.0 each second. When the input is 0.0 the output value doesn't change. And when the input is negative the output decreases over time.

Can also be used to simulate first and second order systems with the input being the differential equation for the rate of change.

Examples: Integrating over the rotation speed give the total rotation angle. Integrating 1.0 gives the total time that the simulation run. Integrating the on ground sensor output gives the total time on ground.

The integral can be reset to 0.0 with an event.

```
        <[integral][LeftEngineFanRotationAngle][]
            <[string8][Input][LeftEngine.N1]>
```

```
                <[float64][Value][0.0]>
            >
```

differentiator

Differentiates the input value over time. When the input increases very quickly the output is very high and returns the rate of change. When the input remains constant the output is zero.

Most airliners have a speed trend arrow on the primary flight displays. The airspeed is differentiated to get the rate of change of the airspeed, the airspeed trend.

```
            <[differentiator][AirspeedTrend][]
                <[string8][Input][AirspeedIndicator.IndicatedAirspeed]>
            >
```

first_order_low_pass

Simulates a first order system where the rate of change is proportional to the difference between the output and the input.

Great low pass filter!

```
d( Output ) / dt = ( Input() - Output() ) / TimeConstant
```

```
            <[first_order_low_pass][TimeDelayed][]
                <[string8][Input][Value.Output]>
                <[float64][TimeConstant][1.0]>
                <[string8][Value][0.0]>
            >
```

delay_clamped

The output slowly follows the input with the set `Speed`. When the input is moving too quickly so that the output can't keep up the difference between them will grow. When the `Threshold` is hit the `Output` will be pulled along very quickly and the `OutputClamped` turns true.

Sort of like a slow dog on a leash that trots along when you walk slowly but as you start running the dog is more or less pulled along.

The output stays clamped within the Input +/- the threshold border.

Useful to detect if the pitch trim is running away, or in other words: the input is constantly changing in one direction so that the output can't keep up. In this case the OutputClamped would be used.

```
// One direction:
// Input moving:     |----x---------| ->>>>   (fast)
// Output              x ->                   (slow)
//
```

```
// Input moving:       |----x---------| ->>>> (fast)
// Output              x ->>>>                (fast)
//
// Other direction
// Input moving:      <<<<- |----x---------|
// Output                            <-x
//
// Input moving:   <<<<- |----x---------|
// Output                          <<<<-x
```

```
            <[delay_clamped][PitchTrimDelayed][]
                <[string8][Input][PitchTrim.Output]>
                <[tmvector2d][Threshold][ -0.03 0.03 ]>
                <[float64][Speed][0.04]>
            >
```

Binary / Discretization

logic_greater

Comparison of the two inputs.

```
if ( Input0() > Input1() ) return 1.0;
else return 0.0;
```

```
            <[logic_greater][Input0GreaterInput1][]
                <[string8][Input0][Input0.Output]>
                <[string8][Input1][Input1.Output]>
            >
```

logic_range

Determines if an input is within a given range.

```
if ( Input0() > Range.min && Input0() < Range.max ) return 1.0;
else return 0.0;
```

```
            <[logic_range][ValueInRange][]
                <[float64][Input][Value.Output]>
                <[float64][Range][ 0.0 1.0 ]>
            >
```

floor

The input is rounded down to the nearest integer.

```
            <[floor][DiscreteValue][]
                <[string8][Input][DoubleValue.Output]>
                <[string8][Threshold][0.001]>
            >
```

ceil

The input is rounded up to the nearest integer.

```
            <[floor][DiscreteValue][]
                <[string8][Input][DoubleValue.Output]>
                <[string8][Threshold][0.001]>
            >
```

discrete_hysteresis

The input may can be dynamically changing. When the input and the current output state differ by more than the set threshold then the output flips to the input value, rounded to the nearest integer.

An example would be a rotary knob that is turned slowly. The output shall be a discrete integer value. But with the hysteresis small vibrations don't cause the output to flicker back and forth.

```
            <[discrete_hysteresis][DiscreteValue][]
                <[string8][Input][DoubleValue.Output]>
                <[string8][Threshold][0.7]>
                <[string8][Events][ DEV0.Trigger ]>
            >
```

variable

Can dynamically be assigned using events.

```
            <[variable][State][]
                <[string8][Value][0.0]>
            >
```

state_frozen

Follows the input as long as it is enabled. When InputEnable drops below 0.5 the output value doesn't change.

```
            <[state_frozen][FollowValueWhenConditionMet][]
                <[string8][Input][Value.Output]>
                <[string8][InputEnable][Condition.Output]>
                <[float64][Value][1.0]>
```

```
>
```

logic_confirm_delay

Same as logic_greater but with an adjustable time delay.

```
<[logic_confirm_delay][FMGC1Powered][]
    <[string8][Input][FMGC1On.Output]>
    <[float64][TimeUp][0.001]>
    <[float64][TimeDown][25.0]>
    <[float64][Threshold][0.5]>
>
```

flasher_rectangle

Periodically turns on and off. The total time in seconds for a cycle is the `Period` and the fraction of that time that the output is active is `FractionActive` between 0.0 and 1.0.

```
<[flasher_rectangle][Flasher][]
    <[float64][Period][1.0]>
    <[float64][FractionActive][0.5]>
>
```

Logic Gates

input_active

Returns 1.0 as long as a button is depressed. Returns 0.0 if not. Incoming messages are filtered with the InputValue (see Inputs.

```
<[input_active][ButtonDepressed][]
    <[string8][Input][Controls.Button]>
    <[float64][InputValue][1.0]>
>
```

input_binary

Binary value (0.0 or 1.0) that can be toggled on and off with an input. Push a button to set it on, push the same button to turn it off. Ignores all inputs when the InputEnable is below 0.5.

`Value` is the initial state.

```
<[input_binary][BinaryState][]
    <[string8][Input][Controls.State]>
```

```
            <[string8][InputEnable][1.0]>
            <[float64][Value][0.0]>
        >
```

input_discrete

Simulated integer state within a certain range. E.g. can be used for the selected menu item, to step through a range setting, digital volume setting, etc.

`Value` is the initial state.

`Toggle` when set to true allows the input to toggle a position on and off. The first press selects the position to on, a second press turns it back off and sets the output to 0.0.

```
        <[input_discrete][NDPilotInformation][]
            <[string8][Input][NavigationDisplayPilot.Information]>
            <[string8][Range][ 0.0 5.0 ]>
            <[float64][Value][5.0]>
            <[bool][Toggle][true]>
        >
```

logic_invert

Negates the input.

```
if ( Input0() < 0.5 ) return 1.0;
else return 0.0;
```

```
        <[logic_invert][NotA][]
            <[string8][Input][A.Output]>
        >
```

logic_equal

Returns true (1.0) when the two inputs are equal (difference very small).

```
        <[logic_equal][AEqualsB][]
            <[string8][Input0][A.Output]>
            <[string8][Input1][B.Output]>
        >
```

logic_set

Returns true (1.0) when the input is equal to any of the values in the set.

```
        <[logic_set][LeftMagnetoLeft][]
```

```
            <[string8][Input][MagnetosLeft.Output]>
            <[string8][Set][2 3 4]>
        >
```

logic_and

When all input are true (above 0.5) the logic_and returns 1.0;

```
        <[logic_and][ABAndC][]
            <[string8][Inputs][ A.Output B.Output C.Output ]>
        >
```

logic_or

When any input is true (greater 0.5) the logic_or returns true

```
        <[logic_or][ABOrC][]
            <[string8][Inputs][ A.Output B.Output C.Output ]>
        >
```

logic_xor

The XOR gate returns 1.0 when the inputs are different (either a true or b true but not both).

```
if( ( Input0() > 0.5 ) != ( Input1() > 0.5 ) ) return 1.0;
else return 0.0;
```

```
        <[logic_xor][ExactlyOneGeneratorOff][]
            <[string8][Input0][LeftGeneratorOff.Output]>
            <[string8][Input1][RightGeneratorOff.Output]>
        >
```

logic_combination

Some combinations are tricky to evaluate with the elements introduced so far. When complex logic is needed for deciding an outcome, e.g. of a multi-failure mode, the logic_combination can be used to save loads of tmd objects.

```
        <[logic_combination][ErrorCases][]
            <[string8][Inputs][ A.Output B.Output C.Output ]>
        >
```

Available combinations are requested with the output functions:

Output	Description
OutputNone	**NOR** - 1.0 if all inputs are below 0.5
OutputAny	**OR** - 1.0 if any input above 0.5
OutputAll	**AND** - 1.0 if all inputs above 0.5
OutputCount	Number of inputs above 0.5
OutputSingle	Exactly one input above 0.5
OutputDual	Exactly two inputs above 0.5
OutputTriple	Exactly three inputs above 0.5
OutputExclusively0	Only the first input above 0.5
OutputExclusively1	Only the second input above 0.5
OutputExclusively2	Only the third input above 0.5
OutputExclusively3	Only the fourth input above 0.5
OutputExclusively4	Only the fifth input above 0.5
OutputExclusively5	Only the sixth input above 0.5
OutputExclusively6	Only the seventh input above 0.5
OutputExclusively7	Only the eighth input above 0.5