qbook - basic classes

amium

Item

Item — (text)
- string Text

Signal — (text [, value, unit, s/C color])
- double Value
- string Unit
- string Color
- fFormat Format

Timer — (text, interval)
- double Interval, Repetitions
- double Min, Max, Step
- Signal Ticks
- virtual void Run(), Abort()
- OnElapesdDelegate -> (Timer t)
- delegate void OnElapsed (Timer t)

Table

Module — (text [, s/C Color])
- Signal Read, Set, Out
- string Status
- int State, Mode, Alert
- string Color
- Dictionary<object> Tags
- object Tag(name [, object tag])

Item::Timer::Module

Pid — (text, t, ks, tu, tg)
- double T, KS, Tu, Tg

Simulation
- Input:Read
- Output:Out

Signalgenerator — (text, signalType, period, min, max)
- SignalType: Sin, Cos, Sawtooth, Rectangle, Triangle, Noise
- double Period, Min, Max
- Output:Out

Item::Timer

Machine — (text, FunctionDelegate resetFunction)
- State: Reset, Resetted, Run, Running, Pause, Paused, Abort, Aborted, Finish, Finished
- Elapsed(Timer t) -> call ‚function'
- string Status
- State State
- FunctionDelegate -> (Machine m, DateTime start)
- void Function(FunctionDelegate function)
- void Reset(), Run(), Pause(), Finish(), Abort()
- Dictionary<Module> Modules
- Module Module(name [, Module module])
- Dictionary<Signal> Signals
- Signal Signal(name [, Signal signal])
- Dictionary<Control> Controls
- Control Control(name [, Control control])
- Dictionary<object> Tags
- object Tag(name [, object tag])

Item

Control — (text [, x, y, w, h])
- OnClickDelegate -> (Control c)
- delegate void OnClick (Control c)
- string Border
- string BackColor

Control::Label
- Format Format

Label::Button — (text, [, w, h | , x, y, w, h,] OnClickDelegate onClick)

Panel
- Dictionary<Control> Controls
- Control Control(name [, Control control])

Widget
- Signal Signal
- Module Module

Gauge
- Axis Axis
- Module Module
- Signal Signal

Chart
- Axis AxisX
- Axis AxisY1
- Dictionary<Signal> Signals
- Signal Signal(name [, Signal signal])

Item::Timer::Machine

UDL.Client — (text, string socket)

```
#-# Initialize:
UDL.Client machine = new UDL.Client("#text#", "127.0.0.1:9001");
machine.Function((m, start) => Idle(m as UDL.Client, start);
machine.Run((m) => Idle(m as UDL.Client));
#-# Code:
async void Idle(UDL.Client uc, DateTime start) {
    ## idle code here ##
}
```

AK.Client — (text, string socket)
- string Transceive(string command [, ref int duration])
- char Fsb
- string Access, Status

```
#-# Initialize:
AK.Client machine = new AK.Client("Client", "127.0.0.1:7700");
machine.Function((m, date) => Idle(m as AK.Client, date);
machine.Run();
#-# Code:
async void Idle(AK.Client akc, DateTime start) {
    ## idle code here ##
}
```

Item::Timer::Machine

AK.Server — (text [, int port | List<int> ports])
- delegate string OnMessageReceived (Server s, int port, string client, char dcb, string command, string channel, string[] parameters) return: string response
- char Dcb, Fsb
- string Access, Status

```
#-# Initialize:
AK.Server machine = new AK.Server("Host", 7700);
// AK.Server machine = new AK.Server("Host", new List<int>() {7700, 7800});
machine.OnMessageReceived = (m, port, client, dcb, command, channel, parameter) => MessageReceived(m, port, client, dcb, command, channel, parameter);
machine.Function((m, date) => Idle(m as AK.Server, date);
machine.Run();

#-# Code:
public void Idle(AK.Server aks, DateTime start) { // every 10ms
    ## idle code here ##
}

string MessageReceived(AK.Server aks, int port, string client, char dcb, string command, string channel, string[] parameters) {
    ## message received code here ##

    return „#response#";
}
```