```
const std = @import("std");

const dds = @import("dds");

// keyboard
const kbd = @import("cursed").kbd;

// panel
const pnl = @import("forms").pnl;
// button
const btn = @import("forms").btn;
// label
const lbl = @import("forms").lbl;
// menu
const mnu = @import("forms").mnu;
// flied
const fld = @import("forms").fld;
// line horizontal
const lnh = @import("forms").lnh;
// line vertival
const lnv = @import("forms").lnv;

// grid
const grd = @import("grid").grd;

// full delete for produc
const forms = @import("forms");

const allocator = std.heap.page_allocator;

// tools utility
const utl = @import("utils");

const deb_Log = @import("logger").openFile; // open  file
const end_Log = @import("logger").closeFile; // close file
const plog = @import("logger").scoped; // print file


//..............................//
// define Ctype JSON
const Ctype = enum { null, bool, integer, float, number_string, string, array, object, decimal_string };

//..............................//
// define BUTTON JSON
//..............................//

const DEFBUTTON = struct { name: []const u8, key: kbd, show: bool, check: bool, title: []const u8 };

const Jbutton = enum { name, key, show, check, title };

//..............................//
// define LABEL JSON
//..............................//

const DEFLABEL = struct { name: []const u8, posx: usize, posy: usize, text: []const u8, title: bool };

const Jlabel = enum { name, posx, posy, text, title };


//..............................//
// define FIELD JSON
//..............................//

pub const DEFFIELD = struct {
    name: []const u8,
    posx: usize,
    posy: usize,
    reftyp: dds.REFTYP,
    width: usize,
    scal: usize,
    requier: bool, // requier or FULL
    protect: bool, // only display
    edtcar: []const u8, // edtcar ex: monnaie

    regex: []const u8, //contrÃ´le regex
    errmsg: []const u8, //message this field

    help: []const u8, //help this field

    text: []const u8,
    zwitch: bool, // CTRUE CFALSE

    procfunc: []const u8, //name proc

    proctask: []const u8, //name proc

    actif: bool,
};

const Jfield = enum {
    name,
    posx,
    posy,
    reftyp,
    width,
    scal,
    text,
    requier,
    protect,
    edtcar,
    errmsg,
    help,
    procfunc,
    proctask
};



//..............................//
// define PANEL JSON
//..............................//
const RPANEL = struct {
    name: []const u8,
    posx: usize,
    posy: usize,
    lines: usize,
    cols: usize,
    cadre: dds.CADRE,
    title: []const u8,
    button: std.ArrayList(DEFBUTTON),
    label: std.ArrayList(DEFLABEL),
    field: std.ArrayList(DEFFIELD)
};

const Jpanel = enum {
    name,
    posx,
    posy,
    lines,
    cols,
    cadre,
    title,
    button,
    label,
    field
};

var ENRG: RPANEL = undefined;

//..............................//
//  string return enum
//..............................//

fn strToEnum(comptime EnumTag: type, vtext: []const u8) EnumTag {
    inline for (@typeInfo(EnumTag).Enum.fields) |f| {
        if (std.mem.eql(u8, f.name, vtext)) return @field(EnumTag, f.name);
    }

    var buffer: [128]u8 = [_]u8{0} ** 128;
    var result = std.fmt.bufPrintZ(buffer[0..], "invalid Text {s} for strToEnum ", .{vtext}) catch unreachable;
    @panic(result);
}




//..............................//
// JSON
//..............................//

const T = struct {
    x: ?std.json.Value,

    pub fn init(self: std.json.Value) T {
        return T{ .x = self };
    }

    pub fn get(self: T, query: []const u8) T {
        if (self.x.?.object.get(query) == null) {
            std.debug.print("ERROR::{s}::", .{"invalid"});
            return T.init(self.x.?);
        }

        return T.init(self.x.?.object.get(query).?);
    }

    pub fn ctrlPack(self: T, Xtype: Ctype) !bool {
        var out = std.ArrayList(u8).init(allocator);
        defer out.deinit();

        switch (self.x.?) {
            .null => {
                if (Xtype != .null) return false;
            },

            .bool => {
                if (Xtype != Ctype.bool) return false;
            },

            .integer => {
                if (Xtype != Ctype.integer) return false;
            },

            .float => {
                if (Xtype != Ctype.float) return false;
            },

            .number_string => {
                if (Xtype != Ctype.number_string) return false;
            },

            .string => {
                if (Xtype != Ctype.string) return false;
                if (Xtype == Ctype.decimal_string)
                    return utl.isDecimalStr(try std.fmt.allocPrint(allocator, "{s}", .{self.x.?.string}));
            },

            .array => {
                if (Xtype != Ctype.array) return false;
            },

            .object => {
                if (Xtype != Ctype.object) return false;
                //try printPack(self,Xtype);
            },
        }

        return true;
    }

    pub fn index(self: T, i: usize) T {
        switch (self.x.?) {
            .array => {
                if (i > self.x.?.array.items.len) {
                    std.debug.print("ERROR::{s}::\n", .{"index out of bounds"});
                    return T.init(self.x.?);
                }
            },
            else => {
                std.debug.print("ERROR::{s}:: {s}\n", .{ "Not array", @tagName(self.x.?) });
                return T.init(self.x.?);
            },
        }
        return T.init(self.x.?.array.items[i]);
    }
};


//..............................//
// DECODEUR
//..............................//

pub fn jsonDecode(my_json: []const u8) !void {
    var val: T = undefined;

    const parsed = try std.json.parseFromSlice(std.json.Value, allocator, my_json, .{});
    defer parsed.deinit();

    std.debug.print("\n", .{});

    const json = T.init(parsed.value);

    _ = try json.ctrlPack(Ctype.object);

    val = json.get("PANEL");

    var nbrPanel = val.x.?.array.items.len;

    var p: usize= 0;

    const Rpanel = std.enums.EnumIndexer(Jpanel);

    const Rbutton = std.enums.EnumIndexer(Jbutton);

    const Rlabel = std.enums.EnumIndexer(Jlabel);

    const Rfield = std.enums.EnumIndexer(Jfield);

    while (p < nbrPanel) : (p += 1) {
        var n: usize = 0; // index

        while (n < Rpanel.count) : (n += 1) {
            var v: usize = 0; // index
            var y: usize = 0; // array len
            var z: usize = 0; // compteur
            var b: usize = 0; // button
            var l: usize = 0; // label
            var f: usize = 0; // field

            switch (Rpanel.keyForIndex(n)) {
                Jpanel.name => {
                    val = json.get("PANEL").index(p).get(@tagName(Rpanel.keyForIndex(n)));

                    if (try val.ctrlPack(Ctype.string))
                        ENRG.name = try std.fmt.allocPrint(allocator, "{s}", .{val.x.?.string})
                    else
                        @panic(try std.fmt.allocPrint(allocator,
                        "Json  Panel err_Field :{s}\n", .{@tagName(Rpanel.keyForIndex(n))}));
                },
                Jpanel.posx => {
                    val = json.get("PANEL").index(p).get(@tagName(Rpanel.keyForIndex(n)));

                    if (try val.ctrlPack(Ctype.integer))
                        ENRG.posx = @intCast(val.x.?.integer)
                    else
                        @panic(try std.fmt.allocPrint(allocator,
                        "Json  err_Field :{s}\n", .{@tagName(Rpanel.keyForIndex(n))}));
                },
                Jpanel.posy => {
```

```
                val = json.get("PANEL").index(p).get(@tagName(Rpanel.keyForIndex(n)));

                if (try val.ctrlPack(Ctype.integer))
                    ENRG.posy = @intCast(val.x.?.integer)
                else
                    @panic(try std.fmt.allocPrint(allocator,
                    "Json  err_Field :{s}\n", .{@tagName(Rpanel.keyForIndex(n))}));
            },
            Jpanel.lines => {
                val = json.get("PANEL").index(p).get(@tagName(Rpanel.keyForIndex(n)));

                if (try val.ctrlPack(Ctype.integer))
                    ENRG.lines = @intCast(val.x.?.integer)
                else
                    @panic(try std.fmt.allocPrint(allocator,
                    "Json  err_Field :{s}\n", .{@tagName(Rpanel.keyForIndex(n))}));
            },
            Jpanel.cols => {
                val = json.get("PANEL").index(p).get(@tagName(Rpanel.keyForIndex(n)));

                if (try val.ctrlPack(Ctype.integer))
                    ENRG.cols = @intCast(val.x.?.integer)
                else
                    @panic(try std.fmt.allocPrint(allocator,
                    "Json  err_Field :{s}\n", .{@tagName(Rpanel.keyForIndex(n))}));
            },
            Jpanel.cadre => {
                val = json.get("PANEL").index(p).get(@tagName(Rpanel.keyForIndex(n)));

                if (try val.ctrlPack(Ctype.string)) {
                    ENRG.cadre = strToEnum(dds.CADRE, val.x.?.string);
                } else @panic(try std.fmt.allocPrint(allocator,
                    "Json  err_Field :{s}\n", .{@tagName(Rpanel.keyForIndex(n))}));
            },
            Jpanel.title => {
                val = json.get("PANEL").index(p).get(@tagName(Rpanel.keyForIndex(n)));

                if (try val.ctrlPack(Ctype.string))
                    ENRG.title = try std.fmt.allocPrint(allocator, "{s}", .{val.x.?.string})
                else
                    @panic(try std.fmt.allocPrint(allocator,
                    "Json  err_Field :{s}\n", .{@tagName(Rpanel.keyForIndex(n))}));
            },
            //=============================================================================
            // BUTTON
            //=============================================================================
            Jpanel.button => {
                val = json.get("PANEL").index(p).get(@tagName(Rpanel.keyForIndex(n)));

                var bt: DEFBUTTON = undefined;
                y = val.x.?.array.items.len;
                z = 0;
                b = 0;

                while (z < y) : (z += 1) {
                    v = 0;
                    while (v < Rbutton.count) : (v += 1) {
                        val = json.get("PANEL").index(p).get("button").index(b)
                            .get(@tagName(Rbutton.keyForIndex(v))
                        );

                        switch (Rbutton.keyForIndex(v)) {
                            Jbutton.name => {
                                if (try val.ctrlPack(Ctype.string))
                                    bt.name = try std.fmt.allocPrint(allocator, "{s}", .{val.x.?.string})
                                else
                                    @panic(try std.fmt.allocPrint(allocator,
                                    "Json  err_Field :{s}.{s}\n", .{
                                        @tagName(Rpanel.keyForIndex(n)), @tagName(Rbutton.keyForIndex(v))
                                    }));
                            },
                            Jbutton.key => {
                                if (try val.ctrlPack(Ctype.string)) {
                                    bt.key = strToEnum(kbd, val.x.?.string);
                                } else @panic(try std.fmt.allocPrint(allocator,
                                    "Json  err_Field :{s}.{s}\n", .{
                                        @tagName(Rpanel.keyForIndex(n)), @tagName(Rbutton.keyForIndex(v))
                                    }));
                            },
                            Jbutton.show => {
                                if (try val.ctrlPack(Ctype.bool))
                                    bt.show = val.x.?.bool
                                else
                                    @panic(try std.fmt.allocPrint(allocator,
                                    "Json  err_Field :{s}.{s}\n", .{
                                        @tagName(Rpanel.keyForIndex(n)), @tagName(Rbutton.keyForIndex(v))
                                    }));
                            },
                            Jbutton.check => {
                                if (try val.ctrlPack(Ctype.bool))
                                    bt.check = val.x.?.bool
                                else
                                    @panic(try std.fmt.allocPrint(allocator,
                                    "Json  err_Field :{s}.{s}\n", .{
                                        @tagName(Rpanel.keyForIndex(n)), @tagName(Rbutton.keyForIndex(v))
                                    }));
                            },
                            Jbutton.title => {
                                if (try val.ctrlPack(Ctype.string))
                                    bt.title = try std.fmt.allocPrint(allocator, "{s}", .{val.x.?.string})
                                else
                                    @panic(try std.fmt.allocPrint(allocator,
                                    "Json  err_Field :{s}.{s}\n", .{
                                        @tagName(Rpanel.keyForIndex(n)), @tagName(Rbutton.keyForIndex(v))
                                    }));

                                ENRG.button.append(bt) catch unreachable;
                            },
                        }
                    }
                    b += 1;
                }
            },
            //===============================================================================
            // LABEL
            //===============================================================================

            Jpanel.label => {
                val = json.get("PANEL").index(p).get(@tagName(Rpanel.keyForIndex(n)));

                var lb: DEFLABEL = undefined;
                y = val.x.?.array.items.len;
                z = 0;
                l = 0;
                while (z < y) : (z += 1) {
                    v = 0;
                    while (v < Rlabel.count) : (v += 1) {
                        val = json.get("PANEL").index(p).get("label").index(l)
                            .get(@tagName(Rlabel.keyForIndex(v))
                        );

                        switch (Rlabel.keyForIndex(v)) {
                            Jlabel.name => {
                                if (try val.ctrlPack(Ctype.string))
                                    lb.name = try std.fmt.allocPrint(allocator, "{s}", .{val.x.?.string})
                                else
                                    @panic(try std.fmt.allocPrint(allocator,
                                        "Json  err_Field :{s}.{s}\n", .{
                                        @tagName(Rpanel.keyForIndex(n)), @tagName(Rlabel.keyForIndex(v))
                                    }));
                            },
                            Jlabel.posx => {
                                if (try val.ctrlPack(Ctype.integer)) {
                                    lb.posx = @intCast(val.x.?.integer);
                                } else @panic(try std.fmt.allocPrint(allocator,
                                    "Json  err_Field :{s}.{s}\n", .{
                                        @tagName(Rpanel.keyForIndex(n)), @tagName(Rbutton.keyForIndex(v))
                                    }));
                            },
                            Jlabel.posy => {
                                if (try val.ctrlPack(Ctype.integer)) {
                                    lb.posy = @intCast(val.x.?.integer);
                                } else @panic(try std.fmt.allocPrint(allocator,
                                    "Json  err_Field :{s}.{s}\n", .{
                                        @tagName(Rpanel.keyForIndex(n)), @tagName(Rbutton.keyForIndex(v))
                                    }));
                            },
                            Jlabel.text => {
                                if (try val.ctrlPack(Ctype.string))
                                    lb.text = try std.fmt.allocPrint(allocator, "{s}", .{val.x.?.string})
                                else
                                    @panic(try std.fmt.allocPrint(allocator,
                                    "Json  err_Field :{s}.{s}\n", .{
                                        @tagName(Rpanel.keyForIndex(n)), @tagName(Rlabel.keyForIndex(v))
                                    }));
                            },
                            Jlabel.title => {
                                if (try val.ctrlPack(Ctype.bool))
                                    lb.title = val.x.?.bool
                                else
                                    @panic(try std.fmt.allocPrint(allocator,
                                    "Json  err_Field :{s}.{s}\n", .{
                                        @tagName(Rpanel.keyForIndex(n)), @tagName(Rbutton.keyForIndex(v))
                                    }));

                                ENRG.label.append(lb) catch unreachable;
                            },
                        }
                    }

                    l += 1;
                }
            },
            //===============================================================================
            // FIELD
            //===============================================================================

            Jpanel.field => {
                val = json.get("PANEL").index(p).get(@tagName(Rpanel.keyForIndex(n)));
                std.debug.print("field: {any}\n\n",.{val.x.?} );

                var sreftyp:[]const u8 = undefined;

                var lf: DEFFIELD = undefined;
                y = val.x.?.array.items.len;
                z = 0;
                f = 0;
                while (z < y) : (z += 1) {
                    v = 0;
                    while (v < Rfield.count) : (v += 1) {
```

```zig

                        val = json.get("PANEL").index(p).get("field").index(f)
                                        .get(@tagName(Rfield.keyForIndex(v))
                                    );

                        switch (Rfield.keyForIndex(v)) {
                            Jfield.name => { if (try val.ctrlPack(Ctype.string))
                                    lf.name = try std.fmt.allocPrint(allocator, "{s}", .{val.x.?.string})
                                else
                                    @panic(try std.fmt.allocPrint(allocator,
                                        "Json  err_Field :{s}.{s}\n", .{
                                        @tagName(Rpanel.keyForIndex(n)), @tagName(Rfield.keyForIndex(v))
                                    }));
                            },

                            Jfield.posx => { if (try val.ctrlPack(Ctype.integer)) {
                                    lf.posx = @intCast(val.x.?.integer);
                                } else @panic(try std.fmt.allocPrint(allocator,
                                    "Json  err_Field :{s}.{s}\n", .{
                                        @tagName(Rpanel.keyForIndex(n)), @tagName(Rbutton.keyForIndex(v))
                                    }));
                            },

                            Jfield.posy => { if (try val.ctrlPack(Ctype.integer)) {
                                    lf.posy = @intCast(val.x.?.integer);
                                } else @panic(try std.fmt.allocPrint(allocator,
                                    "Json  err_Field :{s}.{s}\n", .{
                                        @tagName(Rpanel.keyForIndex(n)), @tagName(Rbutton.keyForIndex(v))
                                    }));
                            },

                            Jfield.reftyp => {
                                    if (try val.ctrlPack(Ctype.string)) {
                                        sreftyp = try std.fmt.allocPrint(allocator, "{s}", .{val.x.?.string});

                                    } else @panic(try std.fmt.allocPrint(allocator,
                                            "Json  err_Field :{s}.{s}\n", .{
                                            @tagName(Rpanel.keyForIndex(n)), @tagName(Rfield.keyForIndex(v))
                                        }));

                                    lf.reftyp = strToEnum(dds.REFTYP ,sreftyp);
                            },

                            Jfield.width => {if (try val.ctrlPack(Ctype.integer)) {
                                    lf.width= @intCast(val.x.?.integer);
                                } else @panic(try std.fmt.allocPrint(allocator,
                                    "Json  err_Field :{s}.{s}\n", .{
                                        @tagName(Rpanel.keyForIndex(n)), @tagName(Rbutton.keyForIndex(v))
                                    }));
                            },

                            Jfield.scal => {if (try val.ctrlPack(Ctype.integer)) {
                                    lf.scal= @intCast(val.x.?.integer);
                                } else @panic(try std.fmt.allocPrint(allocator,
                                    "Json  err_Field :{s}.{s}\n", .{
                                        @tagName(Rpanel.keyForIndex(n)), @tagName(Rbutton.keyForIndex(v))
                                    }));

                            },
                            Jfield.text=>{
                                lf.text="";
                            },

                            Jfield.requier => {
                                if (try val.ctrlPack(Ctype.bool)) {
                                    lf.requier = val.x.?.bool ;
                                } else @panic(try std.fmt.allocPrint(allocator,
                                    "Json  err_Field :{s}.{s}\n", .{
                                        @tagName(Rpanel.keyForIndex(n)), @tagName(Rbutton.keyForIndex(v))
                                    }));
                            },

                            Jfield.protect=> {
                                if (try val.ctrlPack(Ctype.bool)) {
                                    lf.protect= val.x.?.bool ;
                                } else @panic(try std.fmt.allocPrint(allocator,
                                    "Json  err_Field :{s}.{s}\n", .{
                                        @tagName(Rpanel.keyForIndex(n)), @tagName(Rbutton.keyForIndex(v))
                                    }));
                            },

                            Jfield.edtcar => { if (try val.ctrlPack(Ctype.string)) {
                                    lf.edtcar= try std.fmt.allocPrint(allocator, "{s}", .{val.x.?.string});
                                } else @panic(try std.fmt.allocPrint(allocator,
                                        "Json  err_Field :{s}.{s}\n", .{
                                        @tagName(Rpanel.keyForIndex(n)), @tagName(Rfield.keyForIndex(v))
                                    }));
                            },

                            Jfield.errmsg=> { if (try val.ctrlPack(Ctype.string)) {
                                    lf.errmsg= try std.fmt.allocPrint(allocator, "{s}", .{val.x.?.string});
                                } else @panic(try std.fmt.allocPrint(allocator,
                                        "Json  err_Field :{s}.{s}\n", .{
                                        @tagName(Rpanel.keyForIndex(n)), @tagName(Rfield.keyForIndex(v))
                                    }));
                            },

                            Jfield.help=> { if (try val.ctrlPack(Ctype.string)) {
                                    lf.help= try std.fmt.allocPrint(allocator, "{s}", .{val.x.?.string});
                                } else @panic(try std.fmt.allocPrint(allocator,
                                        "Json  err_Field :{s}.{s}\n", .{
                                        @tagName(Rpanel.keyForIndex(n)), @tagName(Rfield.keyForIndex(v))
                                    }));
                            },

                            Jfield.procfunc=> { if (try val.ctrlPack(Ctype.string)) {
                                    lf.procfunc= try std.fmt.allocPrint(allocator, "{s}", .{val.x.?.string});
                                } else @panic(try std.fmt.allocPrint(allocator,
                                        "Json  err_Field :{s}.{s}\n", .{
                                        @tagName(Rpanel.keyForIndex(n)), @tagName(Rfield.keyForIndex(v))
                                    }));
                            },

                            Jfield.proctask => { if (try val.ctrlPack(Ctype.string)) {
                                    lf.proctask = try std.fmt.allocPrint(allocator, "{s}", .{val.x.?.string});
                                } else @panic(try std.fmt.allocPrint(allocator,
                                        "Json  err_Field :{s}.{s}\n", .{
                                        @tagName(Rpanel.keyForIndex(n)), @tagName(Rfield.keyForIndex(v))
                                    }));
                                 ENRG.field.append(lf) catch unreachable;
                            },
                        }
                    }

                    f += 1;
                }
            },

        }
    }
}
}

//..............................//
// Main function
//..............................//
pub fn main() !void {
    var my_file = try std.fs.cwd().openFile("fileJson.txt", .{});
        defer my_file.close();


    const file_size = try my_file.getEndPos();
    var buffer : []u8= allocator.alloc(u8, file_size) catch unreachable ;


    _= try my_file.read(buffer[0..buffer.len]);

    // init arraylist
    ENRG.button = std.ArrayList(DEFBUTTON).init(allocator);
    ENRG.label = std.ArrayList(DEFLABEL).init(allocator);
    ENRG.field = std.ArrayList(DEFFIELD).init(allocator);

    jsonDecode(buffer) catch return;

    deb_Log("zmodlRJson.txt");

    plog(.schema).debug("\nwrite Json", .{});
    plog(.schema).debug("\n{s}\n", .{buffer});
    plog(.schema).debug("\nRead Json", .{});

    plog(.Panel).debug("\n", .{});
    plog(.Panel).debug("{s}", .{ENRG.name});
    plog(.Panel).debug("{d}", .{ENRG.posx});
    plog(.Panel).debug("{d}", .{ENRG.posy});
    plog(.Panel).debug("{}", .{ENRG.cadre});
    plog(.Panel).debug("{s}\n", .{ENRG.title});

    plog(.Button).debug("\n", .{});
    for (ENRG.button.items) |r| {
        plog(.Button).debug("{s}", .{r.name});
        plog(.Button).debug("{any}", .{r.key});
        plog(.Button).debug("{}", .{r.show});
        plog(.Button).debug("{}", .{r.check});
        plog(.Button).debug("{s}\n", .{r.title});
    }

    plog(.Label).debug("\n", .{});
    for (ENRG.label.items) |r| {
        plog(.Label).debug("{s}", .{r.name});
        plog(.Label).debug("{d}", .{r.posx});
        plog(.Label).debug("{d}", .{r.posy});
        plog(.Label).debug("{s}", .{r.text});
        plog(.Label).debug("{}\n", .{r.title});
    }

    plog(.Field).debug("\n", .{});
    for (ENRG.field.items) |r| {
        plog(.Field).debug("{s}", .{r.name});
        plog(.Field).debug("{d}", .{r.posx});
        plog(.Field).debug("{d}", .{r.posy});
        plog(.Field).debug("{s}", .{@tagName(r.reftyp)});
        plog(.Field).debug("\n{d}", .{r.width});
        plog(.Field).debug("{d}", .{r.scal});
        plog(.Field).debug("\n{}", .{r.requier});
        plog(.Field).debug("{}", .{r.protect});
        plog(.Field).debug("{s}\n", .{r.edtcar});
        plog(.Field).debug("{s}\n", .{r.errmsg});
        plog(.Field).debug("{s}\n", .{r.help});
        plog(.Field).debug("{s}\n", .{r.procfunc});
        plog(.Field).debug("{s}\n", .{r.proctask});
    }


    plog(.end).debug("End.\n", .{});

    end_Log();
}
```