```zig
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,
```

```
101:     errmsg,
102:     help,
103:     procfunc,
104:     proctask
105: };
106:
107:
108:
109: //.............................//
110: // define PANEL JSON
111: //.............................//
112: const RPANEL = struct {
113:     name: []const u8,
114:     posx: usize,
115:     posy: usize,
116:     lines: usize,
117:     cols: usize,
118:     cadre: dds.CADRE,
119:     title: []const u8,
120:     button: std.ArrayList(DEFBUTTON),
121:     label: std.ArrayList(DEFLABEL),
122:     field: std.ArrayList(DEFFIELD)
123: };
124:
125: const Jpanel = enum {
126:     name,
127:     posx,
128:     posy,
129:     lines,
130:     cols,
131:     cadre,
132:     title,
133:     button,
134:     label,
135:     field
136: };
137:
138: var ENRG: RPANEL = undefined;
139:
140: //.............................//
141: //  string return enum
142: //.............................//
143:
144: fn strToEnum(comptime EnumTag: type, vtext: []const u8) EnumTag {
145:     inline for (@typeInfo(EnumTag).Enum.fields) |f| {
146:         if (std.mem.eql(u8, f.name, vtext)) return @field(EnumTag, f.name);
147:     }
148:
149:     var buffer: [128]u8 = [_]u8{0} ** 128;
150:     var result = std.fmt.bufPrintZ(buffer[0..], "invalid Text {s} for strToEnum ", .{vtext}) catch unreachable;
151:     @panic(result);
152: }
153:
154:
155:
156:
157:
158: //.............................//
159: // JSON
160: //.............................//
161:
162: const T = struct {
163:     x: ?std.json.Value,
164:
165:     pub fn init(self: std.json.Value) T {
166:         return T{ .x = self };
167:     }
168:
169:     pub fn get(self: T, query: []const u8) T {
170:         if (self.x.?.object.get(query) == null) {
171:             std.debug.print("ERROR::{s}::", .{"invalid"});
172:             return T.init(self.x.?);
173:         }
174:
175:         return T.init(self.x.?.object.get(query).?);
176:     }
177:
178:     pub fn ctrlPack(self: T, Xtype: Ctype) !bool {
179:         var out = std.ArrayList(u8).init(allocator);
180:         defer out.deinit();
181:
182:         switch (self.x.?) {
183:             .null => {
184:                 if (Xtype != .null) return false;
185:             },
186:
187:             .bool => {
188:                 if (Xtype != Ctype.bool) return false;
189:             },
190:
191:             .integer => {
192:                 if (Xtype != Ctype.integer) return false;
193:             },
194:
195:             .float => {
196:                 if (Xtype != Ctype.float) return false;
197:             },
198:
199:             .number_string => {
200:                 if (Xtype != Ctype.number_string) return false;
```

```
201:             },
202:
203:             .string => {
204:                 if (Xtype != Ctype.string) return false;
205:                 if (Xtype == Ctype.decimal_string)
206:                     return utl.isDecimalStr(try std.fmt.allocPrint(allocator, "{s}", .{self.x.?.string}));
207:             },
208:
209:             .array => {
210:                 if (Xtype != Ctype.array) return false;
211:             },
212:
213:             .object => {
214:                 if (Xtype != Ctype.object) return false;
215:                 //try printPack(self,Xtype);
216:             },
217:         }
218:
219:         return true;
220:     }
221:
222:     pub fn index(self: T, i: usize) T {
223:         switch (self.x.?) {
224:             .array => {
225:                 if (i > self.x.?.array.items.len) {
226:                     std.debug.print("ERROR::{s}::\n", .{"index out of bounds"});
227:                     return T.init(self.x.?);
228:                 }
229:             },
230:             else => {
231:                 std.debug.print("ERROR::{s}:: {s}\n", .{ "Not array", @tagName(self.x.?) });
232:                 return T.init(self.x.?);
233:             },
234:         }
235:         return T.init(self.x.?.array.items[i]);
236:     }
237: };
238:
239:
240: //.............................//
241: // DECODEUR
242: //.............................//
243:
244: pub fn jsonDecode(my_json: []const u8) !void {
245:     var val: T = undefined;
246:
247:     const parsed = try std.json.parseFromSlice(std.json.Value, allocator, my_json, .{});
248:     defer parsed.deinit();
249:
250:     std.debug.print("\n", .{});
251:
252:     const json = T.init(parsed.value);
253:
254:     _ = try json.ctrlPack(Ctype.object);
255:
256:     val = json.get("PANEL");
257:
258:     var nbrPanel = val.x.?.array.items.len;
259:
260:     var p: usize= 0;
261:
262:     const Rpanel = std.enums.EnumIndexer(Jpanel);
263:
264:     const Rbutton = std.enums.EnumIndexer(Jbutton);
265:
266:     const Rlabel = std.enums.EnumIndexer(Jlabel);
267:
268:     const Rfield = std.enums.EnumIndexer(Jfield);
269:
270:     while (p < nbrPanel) : (p += 1) {
271:         var n: usize = 0; // index
272:
273:         while (n < Rpanel.count) : (n += 1) {
274:             var v: usize = 0; // index
275:             var y: usize = 0; // array len
276:             var z: usize = 0; // compteur
277:             var b: usize = 0; // button
278:             var l: usize = 0; // label
279:             var f: usize = 0; // field
280:
281:             switch (Rpanel.keyForIndex(n)) {
282:                 Jpanel.name => {
283:                     val = json.get("PANEL").index(p).get(@tagName(Rpanel.keyForIndex(n)));
284:
285:                     if (try val.ctrlPack(Ctype.string))
286:                         ENRG.name = try std.fmt.allocPrint(allocator, "{s}", .{val.x.?.string})
287:                     else
288:                         @panic(try std.fmt.allocPrint(allocator,
289:                         "Json  Panel err_Field :{s}\n", .{@tagName(Rpanel.keyForIndex(n))}));
290:                 },
291:                 Jpanel.posx => {
292:                     val = json.get("PANEL").index(p).get(@tagName(Rpanel.keyForIndex(n)));
293:
294:                     if (try val.ctrlPack(Ctype.integer))
295:                         ENRG.posx = @intCast(val.x.?.integer)
296:                     else
297:                         @panic(try std.fmt.allocPrint(allocator,
298:                         "Json  err_Field :{s}\n", .{@tagName(Rpanel.keyForIndex(n))}));
299:                 },
300:                 Jpanel.posy => {
```

```
301:                     val = json.get("PANEL").index(p).get(@tagName(Rpanel.keyForIndex(n)));
302:
303:                     if (try val.ctrlPack(Ctype.integer))
304:                         ENRG.posy = @intCast(val.x.?.integer)
305:                     else
306:                         @panic(try std.fmt.allocPrint(allocator,
307:                         "Json  err_Field :{s}\n", .{@tagName(Rpanel.keyForIndex(n))}));
308:                 },
309:                 Jpanel.lines => {
310:                     val = json.get("PANEL").index(p).get(@tagName(Rpanel.keyForIndex(n)));
311:
312:                     if (try val.ctrlPack(Ctype.integer))
313:                         ENRG.lines = @intCast(val.x.?.integer)
314:                     else
315:                         @panic(try std.fmt.allocPrint(allocator,
316:                         "Json  err_Field :{s}\n", .{@tagName(Rpanel.keyForIndex(n))}));
317:                 },
318:                 Jpanel.cols => {
319:                     val = json.get("PANEL").index(p).get(@tagName(Rpanel.keyForIndex(n)));
320:
321:                     if (try val.ctrlPack(Ctype.integer))
322:                         ENRG.cols = @intCast(val.x.?.integer)
323:                     else
324:                         @panic(try std.fmt.allocPrint(allocator,
325:                         "Json  err_Field :{s}\n", .{@tagName(Rpanel.keyForIndex(n))}));
326:                 },
327:                 Jpanel.cadre => {
328:                     val = json.get("PANEL").index(p).get(@tagName(Rpanel.keyForIndex(n)));
329:
330:                     if (try val.ctrlPack(Ctype.string)) {
331:                         ENRG.cadre = strToEnum(dds.CADRE, val.x.?.string);
332:                     } else @panic(try std.fmt.allocPrint(allocator,
333:                         "Json  err_Field :{s}\n", .{@tagName(Rpanel.keyForIndex(n))}));
334:                 },
335:                 Jpanel.title => {
336:                     val = json.get("PANEL").index(p).get(@tagName(Rpanel.keyForIndex(n)));
337:
338:                     if (try val.ctrlPack(Ctype.string))
339:                         ENRG.title = try std.fmt.allocPrint(allocator, "{s}", .{val.x.?.string})
340:                     else
341:                         @panic(try std.fmt.allocPrint(allocator,
342:                         "Json  err_Field :{s}\n", .{@tagName(Rpanel.keyForIndex(n))}));
343:                 },
344:                 //=============================================================================
345:                 // BUTTON
346:                 //=============================================================================
347:                 Jpanel.button => {
348:                     val = json.get("PANEL").index(p).get(@tagName(Rpanel.keyForIndex(n)));
349:
350:                     var bt: DEFBUTTON = undefined;
351:                     y = val.x.?.array.items.len;
352:                     z = 0;
353:                     b = 0;
354:
355:                     while (z < y) : (z += 1) {
356:                         v = 0;
357:                         while (v < Rbutton.count) : (v += 1) {
358:                             val = json.get("PANEL").index(p).get("button").index(b)
359:                                 .get(@tagName(Rbutton.keyForIndex(v))
360:                             );
361:
362:                             switch (Rbutton.keyForIndex(v)) {
363:                                 Jbutton.name => {
364:                                     if (try val.ctrlPack(Ctype.string))
365:                                         bt.name = try std.fmt.allocPrint(allocator, "{s}", .{val.x.?.string})
366:                                     else
367:                                         @panic(try std.fmt.allocPrint(allocator,
368:                                         "Json  err_Field :{s}.{s}\n", .{
369:                                             @tagName(Rpanel.keyForIndex(n)), @tagName(Rbutton.keyForIndex(v))
370:                                         }));
371:                                 },
372:                                 Jbutton.key => {
373:                                     if (try val.ctrlPack(Ctype.string)) {
374:                                         bt.key = strToEnum(kbd, val.x.?.string);
375:                                     } else @panic(try std.fmt.allocPrint(allocator,
376:                                         "Json  err_Field :{s}.{s}\n", .{
377:                                             @tagName(Rpanel.keyForIndex(n)), @tagName(Rbutton.keyForIndex(v))
378:                                         }));
379:                                 },
380:                                 Jbutton.show => {
381:                                     if (try val.ctrlPack(Ctype.bool))
382:                                         bt.show = val.x.?.bool
383:                                     else
384:                                         @panic(try std.fmt.allocPrint(allocator,
385:                                         "Json  err_Field :{s}.{s}\n", .{
386:                                             @tagName(Rpanel.keyForIndex(n)), @tagName(Rbutton.keyForIndex(v))
387:                                         }));
388:                                 },
389:                                 Jbutton.check => {
390:                                     if (try val.ctrlPack(Ctype.bool))
391:                                         bt.check = val.x.?.bool
392:                                     else
393:                                         @panic(try std.fmt.allocPrint(allocator,
394:                                         "Json  err_Field :{s}.{s}\n", .{
395:                                             @tagName(Rpanel.keyForIndex(n)), @tagName(Rbutton.keyForIndex(v))
396:                                         }));
397:                                 },
398:                                 Jbutton.title => {
399:                                     if (try val.ctrlPack(Ctype.string))
400:                                         bt.title = try std.fmt.allocPrint(allocator, "{s}", .{val.x.?.string})
```

```
401:                                 else
402:                                     @panic(try std.fmt.allocPrint(allocator,
403:                                     "Json  err_Field :{s}.{s}\n", .{
404:                                         @tagName(Rpanel.keyForIndex(n)), @tagName(Rbutton.keyForIndex(v))
405:                                     }));
406:
407:                                 ENRG.button.append(bt) catch unreachable;
408:                             },
409:                         }
410:                     }
411:                     b += 1;
412:                 }
413:             },
414:             //=============================================================================
415:             // LABEL
416:             //=============================================================================
417:
418:             Jpanel.label => {
419:                 val = json.get("PANEL").index(p).get(@tagName(Rpanel.keyForIndex(n)));
420:
421:                 var lb: DEFLABEL = undefined;
422:                 y = val.x.?.array.items.len;
423:                 z = 0;
424:                 l = 0;
425:                 while (z < y) : (z += 1) {
426:                     v = 0;
427:                     while (v < Rlabel.count) : (v += 1) {
428:                         val = json.get("PANEL").index(p).get("label").index(l)
429:                             .get(@tagName(Rlabel.keyForIndex(v))
430:                         );
431:
432:                         switch (Rlabel.keyForIndex(v)) {
433:                             Jlabel.name => {
434:                                 if (try val.ctrlPack(Ctype.string))
435:                                     lb.name = try std.fmt.allocPrint(allocator, "{s}", .{val.x.?.string})
436:                                 else
437:                                     @panic(try std.fmt.allocPrint(allocator,
438:                                         "Json  err_Field :{s}.{s}\n", .{
439:                                         @tagName(Rpanel.keyForIndex(n)), @tagName(Rlabel.keyForIndex(v))
440:                                     }));
441:                             },
442:                             Jlabel.posx => {
443:                                 if (try val.ctrlPack(Ctype.integer)) {
444:                                     lb.posx = @intCast(val.x.?.integer);
445:                                 } else @panic(try std.fmt.allocPrint(allocator,
446:                                     "Json  err_Field :{s}.{s}\n", .{
447:                                         @tagName(Rpanel.keyForIndex(n)), @tagName(Rbutton.keyForIndex(v))
448:                                     }));
449:                             },
450:                             Jlabel.posy => {
451:                                 if (try val.ctrlPack(Ctype.integer)) {
452:                                     lb.posy = @intCast(val.x.?.integer);
453:                                 } else @panic(try std.fmt.allocPrint(allocator,
454:                                     "Json  err_Field :{s}.{s}\n", .{
455:                                         @tagName(Rpanel.keyForIndex(n)), @tagName(Rbutton.keyForIndex(v))
456:                                     }));
457:                             },
458:                             Jlabel.text => {
459:                                 if (try val.ctrlPack(Ctype.string))
460:                                     lb.text = try std.fmt.allocPrint(allocator, "{s}", .{val.x.?.string})
461:                                 else
462:                                     @panic(try std.fmt.allocPrint(allocator,
463:                                     "Json  err_Field :{s}.{s}\n", .{
464:                                         @tagName(Rpanel.keyForIndex(n)), @tagName(Rlabel.keyForIndex(v))
465:                                     }));
466:                             },
467:                             Jlabel.title => {
468:                                 if (try val.ctrlPack(Ctype.bool))
469:                                     lb.title = val.x.?.bool
470:                                 else
471:                                     @panic(try std.fmt.allocPrint(allocator,
472:                                     "Json  err_Field :{s}.{s}\n", .{
473:                                         @tagName(Rpanel.keyForIndex(n)), @tagName(Rbutton.keyForIndex(v))
474:                                     }));
475:
476:                                 ENRG.label.append(lb) catch unreachable;
477:                             },
478:                         }
479:                     }
480:
481:                     l += 1;
482:                 }
483:             },
484:             //=============================================================================
485:             // FIELD
486:             //=============================================================================
487:
488:             Jpanel.field => {
489:                 val = json.get("PANEL").index(p).get(@tagName(Rpanel.keyForIndex(n)));
490:                 std.debug.print("field: {any}\n\n",.{val.x.?} );
491:
492:                 var sreftyp:[]const u8 = undefined;
493:
494:                 var lf: DEFFIELD = undefined;
495:                 y = val.x.?.array.items.len;
496:                 z = 0;
497:                 f = 0;
498:                 while (z < y) : (z += 1) {
499:                     v = 0;
500:                     while (v < Rfield.count) : (v += 1) {
```

```
501:
502:                                val = json.get("PANEL").index(p).get("field").index(f)
503:                                        .get(@tagName(Rfield.keyForIndex(v))
504:                                    );
505:
506:                                switch (Rfield.keyForIndex(v)) {
507:                                    Jfield.name => { if (try val.ctrlPack(Ctype.string))
508:                                            lf.name = try std.fmt.allocPrint(allocator, "{s}", .{val.x.?.string})
509:                                        else
510:                                            @panic(try std.fmt.allocPrint(allocator,
511:                                                "Json  err_Field :{s}.{s}\n", .{
512:                                                @tagName(Rpanel.keyForIndex(n)), @tagName(Rfield.keyForIndex(v))
513:                                            }));
514:                                    },
515:
516:                                    Jfield.posx => { if (try val.ctrlPack(Ctype.integer)) {
517:                                            lf.posx = @intCast(val.x.?.integer);
518:                                        } else @panic(try std.fmt.allocPrint(allocator,
519:                                            "Json  err_Field :{s}.{s}\n", .{
520:                                                @tagName(Rpanel.keyForIndex(n)), @tagName(Rbutton.keyForIndex(v))
521:                                            }));
522:                                    },
523:
524:                                    Jfield.posy => { if (try val.ctrlPack(Ctype.integer)) {
525:                                            lf.posy = @intCast(val.x.?.integer);
526:                                        } else @panic(try std.fmt.allocPrint(allocator,
527:                                            "Json  err_Field :{s}.{s}\n", .{
528:                                                @tagName(Rpanel.keyForIndex(n)), @tagName(Rbutton.keyForIndex(v))
529:                                            }));
530:                                    },
531:
532:                                    Jfield.reftyp => {
533:                                            if (try val.ctrlPack(Ctype.string)) {
534:                                                sreftyp = try std.fmt.allocPrint(allocator, "{s}", .{val.x.?.string});
535:
536:                                            } else @panic(try std.fmt.allocPrint(allocator,
537:                                                    "Json  err_Field :{s}.{s}\n", .{
538:                                                    @tagName(Rpanel.keyForIndex(n)), @tagName(Rfield.keyForIndex(v))
539:                                                }));
540:
541:                                            lf.reftyp = strToEnum(dds.REFTYP ,sreftyp);
542:                                    },
543:
544:                                    Jfield.width => {if (try val.ctrlPack(Ctype.integer)) {
545:                                            lf.width= @intCast(val.x.?.integer);
546:                                        } else @panic(try std.fmt.allocPrint(allocator,
547:                                            "Json  err_Field :{s}.{s}\n", .{
548:                                                @tagName(Rpanel.keyForIndex(n)), @tagName(Rbutton.keyForIndex(v))
549:                                            }));
550:                                    },
551:
552:                                    Jfield.scal => {if (try val.ctrlPack(Ctype.integer)) {
553:                                            lf.scal= @intCast(val.x.?.integer);
554:                                        } else @panic(try std.fmt.allocPrint(allocator,
555:                                            "Json  err_Field :{s}.{s}\n", .{
556:                                                @tagName(Rpanel.keyForIndex(n)), @tagName(Rbutton.keyForIndex(v))
557:                                            }));
558:
559:                                    },
560:                                    Jfield.text=>{
561:                                        lf.text="";
562:                                    },
563:
564:                                    Jfield.requier => {
565:                                        if (try val.ctrlPack(Ctype.bool)) {
566:                                            lf.requier = val.x.?.bool ;
567:                                        } else @panic(try std.fmt.allocPrint(allocator,
568:                                            "Json  err_Field :{s}.{s}\n", .{
569:                                                @tagName(Rpanel.keyForIndex(n)), @tagName(Rbutton.keyForIndex(v))
570:                                            }));
571:                                    },
572:
573:                                    Jfield.protect=> {
574:                                        if (try val.ctrlPack(Ctype.bool)) {
575:                                            lf.protect= val.x.?.bool ;
576:                                        } else @panic(try std.fmt.allocPrint(allocator,
577:                                            "Json  err_Field :{s}.{s}\n", .{
578:                                                @tagName(Rpanel.keyForIndex(n)), @tagName(Rbutton.keyForIndex(v))
579:                                            }));
580:                                    },
581:
582:                                    Jfield.edtcar => { if (try val.ctrlPack(Ctype.string)) {
583:                                            lf.edtcar= try std.fmt.allocPrint(allocator, "{s}", .{val.x.?.string});
584:                                        } else @panic(try std.fmt.allocPrint(allocator,
585:                                                "Json  err_Field :{s}.{s}\n", .{
586:                                                @tagName(Rpanel.keyForIndex(n)), @tagName(Rfield.keyForIndex(v))
587:                                            }));
588:                                    },
589:
590:                                    Jfield.errmsg=> { if (try val.ctrlPack(Ctype.string)) {
591:                                            lf.errmsg= try std.fmt.allocPrint(allocator, "{s}", .{val.x.?.string});
592:                                        } else @panic(try std.fmt.allocPrint(allocator,
593:                                                "Json  err_Field :{s}.{s}\n", .{
594:                                                @tagName(Rpanel.keyForIndex(n)), @tagName(Rfield.keyForIndex(v))
595:                                            }));
596:                                    },
597:
598:                                    Jfield.help=> { if (try val.ctrlPack(Ctype.string)) {
599:                                            lf.help= try std.fmt.allocPrint(allocator, "{s}", .{val.x.?.string});
600:                                        } else @panic(try std.fmt.allocPrint(allocator,
```

```
601:                                         "Json  err_Field :{s}.{s}\n", .{
602:                                         @tagName(Rpanel.keyForIndex(n)), @tagName(Rfield.keyForIndex(v))
603:                                     }));
604:                                 },
605:
606:                                 Jfield.procfunc=> { if (try val.ctrlPack(Ctype.string)) {
607:                                         lf.procfunc= try std.fmt.allocPrint(allocator, "{s}", .{val.x.?.string});
608:                                     } else @panic(try std.fmt.allocPrint(allocator,
609:                                         "Json  err_Field :{s}.{s}\n", .{
610:                                         @tagName(Rpanel.keyForIndex(n)), @tagName(Rfield.keyForIndex(v))
611:                                     }));
612:                                 },
613:
614:                                 Jfield.proctask => { if (try val.ctrlPack(Ctype.string)) {
615:                                         lf.proctask = try std.fmt.allocPrint(allocator, "{s}", .{val.x.?.string});
616:                                     } else @panic(try std.fmt.allocPrint(allocator,
617:                                         "Json  err_Field :{s}.{s}\n", .{
618:                                         @tagName(Rpanel.keyForIndex(n)), @tagName(Rfield.keyForIndex(v))
619:                                     }));
620:                                  ENRG.field.append(lf) catch unreachable;
621:                                 },
622:                             }
623:                         }
624:
625:                         f += 1;
626:                     }
627:                 },
628:
629:             }
630:         }
631:     }
632: }
633:
634: //.............................//
635: // Main function
636: //.............................//
637: pub fn main() !void {
638:     var my_file = try std.fs.cwd().openFile("fileJson.txt", .{});
639:         defer my_file.close();
640:
641:
642:     const file_size = try my_file.getEndPos();
643:     var buffer : []u8= allocator.alloc(u8, file_size) catch unreachable ;
644:
645:
646:     _= try my_file.read(buffer[0..buffer.len]);
647:
648:     // init arraylist
649:     ENRG.button = std.ArrayList(DEFBUTTON).init(allocator);
650:     ENRG.label = std.ArrayList(DEFLABEL).init(allocator);
651:     ENRG.field = std.ArrayList(DEFFIELD).init(allocator);
652:
653:     jsonDecode(buffer) catch return;
654:
655:     deb_Log("zmodlRJson.txt");
656:
657:     plog(.schema).debug("\nwrite Json", .{});
658:     plog(.schema).debug("\n{s}\n", .{buffer});
659:     plog(.schema).debug("\nRead Json", .{});
660:
661:     plog(.Panel).debug("\n", .{});
662:     plog(.Panel).debug("{s}", .{ENRG.name});
663:     plog(.Panel).debug("{d}", .{ENRG.posx});
664:     plog(.Panel).debug("{d}", .{ENRG.posy});
665:     plog(.Panel).debug("{}", .{ENRG.cadre});
666:     plog(.Panel).debug("{s}\n", .{ENRG.title});
667:
668:     plog(.Button).debug("\n", .{});
669:     for (ENRG.button.items) |r| {
670:         plog(.Button).debug("{s}", .{r.name});
671:         plog(.Button).debug("{any}", .{r.key});
672:         plog(.Button).debug("{}", .{r.show});
673:         plog(.Button).debug("{}", .{r.check});
674:         plog(.Button).debug("{s}\n", .{r.title});
675:     }
676:
677:     plog(.Label).debug("\n", .{});
678:     for (ENRG.label.items) |r| {
679:         plog(.Label).debug("{s}", .{r.name});
680:         plog(.Label).debug("{d}", .{r.posx});
681:         plog(.Label).debug("{d}", .{r.posy});
682:         plog(.Label).debug("{s}", .{r.text});
683:         plog(.Label).debug("{}\n", .{r.title});
684:     }
685:
686:     plog(.Field).debug("\n", .{});
687:     for (ENRG.field.items) |r| {
688:         plog(.Field).debug("{s}", .{r.name});
689:         plog(.Field).debug("{d}", .{r.posx});
690:         plog(.Field).debug("{d}", .{r.posy});
691:         plog(.Field).debug("{s}", .{@tagName(r.reftyp)});
692:         plog(.Field).debug("\n{d}", .{r.width});
693:         plog(.Field).debug("{d}", .{r.scal});
694:         plog(.Field).debug("\n{}", .{r.requier});
695:         plog(.Field).debug("{}", .{r.protect});
696:         plog(.Field).debug("{s}\n", .{r.edtcar});
697:         plog(.Field).debug("{s}\n", .{r.errmsg});
698:         plog(.Field).debug("{s}\n", .{r.help});
699:         plog(.Field).debug("{s}\n", .{r.procfunc});
700:         plog(.Field).debug("{s}\n", .{r.proctask});
```

```
701:     }
702:
703:
704:     plog(.end).debug("End.\n", .{});
705:
706:     end_Log();
707: }
```