

```
1: const std = @import("std");
2:
3: const dds = @import("dds");
4:
5: // keyboard
6: const kbd = @import("cursed").kbd;
7:
8: // panel
9: const pnl = @import("forms").pnl;
10: // button
11: const btn = @import("forms").btn;
12: // label
13: const lbl = @import("forms").lbl;
14: // menu
15: const mnu = @import("forms").mnu;
16: // flied
17: const fld = @import("forms").fld;
18: // line horizontal
19: const lnh = @import("forms").lnh;
20: // line vertival
21: const lnv = @import("forms").lnv;
22:
23: // grid
24: const grd = @import("grid").grd;
25:
26: // full delete for produc
27: const forms = @import("forms");
28:
29: const allocator = std.heap.page_allocator;
30:
31: // tools utility
32: const utl = @import("utils");
33:
34: const deb_Log = @import("logger").openFile; // open file
35: const end_Log = @import("logger").closeFile; // close file
36: const plog = @import("logger").scoped; // print file
37:
38:
39: //.....//
40: // define CType JSON
41: const CType = enum { null, bool, integer, float, number_string, string, array, object, decimal_string };
42:
43: //.....//
44: // define BUTTON JSON
45: //.....//
46:
```

```
47: const DEFBUTTON = struct {key: kbd, show: bool, check: bool, title: []const u8 };
48:
49: const Jbutton = enum {key, show, check, title };
50:
51: //.....//
52: // define LABEL JSON
53: //.....//
54:
55: const DEFLABEL = struct { name: []const u8, posx: usize, posy: usize, text: []const u8, title: bool };
56:
57: const Jlabel = enum { name, posx, posy, text, title };
58:
59:
60: //.....//
61: // define FIELD JSON
62: //.....//
63:
64: pub const DEFFIELD = struct {
65:     name: []const u8,
66:     posx: usize,
67:     posy: usize,
68:     reftyp: dds.REFTYP,
69:     width: usize,
70:     scal: usize,
71:     requier: bool, // requier or FULL
72:     protect: bool, // only display
73:     edtcarr: []const u8, // edtcarr ex: monnaie
74:
75:     regex: []const u8, //contrÃ´le regex
76:     errmsg: []const u8, //message this field
77:
78:     help: []const u8, //help this field
79:
80:     text: []const u8,
81:     zwitch: bool, // CTRUE CFALSE
82:
83:     procfunc: []const u8, //name proc
84:
85:     proctask: []const u8, //name proc
86:
87:     actif: bool,
88: };
89:
90: const Jfield = enum {
91:     name,
92:     posx,
```

```
93:     posy,
94:     reftyp,
95:     width,
96:     scal,
97:     text,
98:     zwitch,
99:     requier,
100:    protect,
101:    edtcar,
102:    errmsg,
103:    help,
104:    procfunc,
105:    proctask
106: };
107:
108:
109:
110: //.....//
111: // define PANEL JSON
112: //.....//
113: const RPANEL = struct {
114:     name: []const u8,
115:     posX: usize,
116:     posY: usize,
117:     lines: usize,
118:     cols: usize,
119:     cadre: dds.CADRE,
120:     title: []const u8,
121:     button: std.ArrayList(DEFBUTTON),
122:     label: std.ArrayList(DEFLABEL),
123:     field: std.ArrayList(DEFFIELD)
124: };
125:
126: const Jpanel = enum {
127:     name,
128:     posX,
129:     posY,
130:     lines,
131:     cols,
132:     cadre,
133:     title,
134:     button,
135:     label,
136:     field
137: };
138:
```

```
139: var ENRG = std.ArrayList(RPANEL).init(allocator);
140:
141: //.....//
142: // string return enum
143: //.....//
144:
145: fn strToEnum(comptime EnumTag: type, vtext: []const u8) EnumTag {
146:     inline for (@TypeInfo(EnumTag).Enum.fields) |f| {
147:         if (std.mem.eql(u8, f.name, vtext)) return @field(EnumTag, f.name);
148:     }
149:
150:     var buffer: [128]u8 = [_]u8{0} ** 128;
151:     var result = std.fmt.bufPrintZ(buffer[0..], "invalid Text {s} for strToEnum ", .{vtext}) catch unreachable;
152:     @panic(result);
153: }
154:
155:
156:
157:
158:
159: //.....//
160: // JSON
161: //.....//
162:
163: const T = struct {
164:     x: ?std.json.Value,
165:     var err : bool = false ;
166:
167:
168:     pub fn init(self: std.json.Value) T {
169:         return T{ .x = self };
170:     }
171:
172:
173:
174:     pub fn get(self: T, query: []const u8) T {
175:         err= false;
176:
177:         if (self.x?.object.get(query) == null) {
178:             std.debug.print("ERROR::{s}::{s}\n\n", .{"invalid", query});
179:             err= true;
180:             return T.init(self.x?);
181:         }
182:
183:         return T.init(self.x?.object.get(query).?);
184:     }
```

```
185:
186:
187:
188:     pub fn ctrlPack(self: T, Xtype: Ctype) bool {
189:         var out = std.ArrayList(u8).init(allocator);
190:         defer out.deinit();
191:
192:         switch (self.x.?) {
193:             .null => {
194:                 if (Xtype != .null) return false;
195:             },
196:
197:             .bool => {
198:                 if (Xtype != Ctype.bool) return false;
199:             },
200:
201:             .integer => {
202:                 if (Xtype != Ctype.integer) return false;
203:             },
204:
205:             .float => {
206:                 if (Xtype != Ctype.float) return false;
207:             },
208:
209:             .number_string => {
210:                 if (Xtype != Ctype.number_string) return false;
211:             },
212:
213:             .string => {
214:                 if (Xtype != Ctype.string) return false;
215:                 if (Xtype == Ctype.decimal_string)
216:                     return utl.isDecimalStr( std.fmt.allocPrint(allocator, "{s}", .{self.x.?.string}) catch unreachable);
217:             },
218:
219:             .array => {
220:                 if (Xtype != Ctype.array) return false;
221:             },
222:
223:             .object => {
224:                 if (Xtype != Ctype.object) return false;
225:                 //try printPack(self, Xtype);
226:             },
227:         }
228:
229:         return true;
230:     }
```

```
231:
232:
233:
234:     pub fn index(self: T, i: usize) T {
235:
236:         err= false;
237:         switch (self.x.?) {
238:             .array => {
239:                 if (i > self.x?.array.items.len) {
240:                     std.debug.print("ERROR::{s}::\n\n", .{"index out of bounds"});
241:                     err = true;
242:                     return T.init(self.x.?);
243:                 }
244:             },
245:             else => {
246:                 std.debug.print("ERROR::{s}:: {s}\n\n", .{ "Not array", @tagName(self.x.?) });
247:                 err = true;
248:                 return T.init(self.x.?);
249:             },
250:         }
251:         return T.init(self.x?.array.items[i]);
252:     }
253: };
254:
255:
256: //.....//
257: // DECODEUR
258: //.....//
259:
260: pub fn jsonDecode(my_json: []const u8) !void {
261:     var val: T = undefined;
262:
263:     const parsed = try std.json.parseFromSlice(std.json.Value, allocator, my_json, .{});
264:     defer parsed.deinit();
265:
266:     std.debug.print("\n", .{});
267:
268:     const json = T.init(parsed.value);
269:
270:
271:     val = json.get("PANEL");
272:
273:     var nbrPanel = val.x?.array.items.len;
274:
275:     var p: usize= 0;
276:
```

```
277: const Rpanel = std.enums.EnumIndexer(Jpanel);
278:
279: const Rbutton = std.enums.EnumIndexer(Jbutton);
280:
281: const Rlabel = std.enums.EnumIndexer(Jlabel);
282:
283: const Rfield = std.enums.EnumIndexer(Jfield);
284:
285: while (p < nbrPanel) : (p += 1) {
286:     var n: usize = 0; // index
287:
288:     ENRG.append(RPANEL{
289:         .name="",
290:         .posx=0,
291:         .posy=0,
292:         .lines=0,
293:         .cols=0,
294:         .cadre=dds.CADRE.line0,
295:         .title="",
296:         .button=std.ArrayList(DEFBUTTON).init(allocator),
297:         .label=std.ArrayList(DEFLABEL).init(allocator),
298:         .field=std.ArrayList(DEFFIELD).init(allocator)
299:     }) catch unreachable;
300:
301:     while (n < Rpanel.count) : (n += 1) {
302:         var v: usize = 0; // index
303:         var y: usize = 0; // array len
304:         var z: usize = 0; // compteur
305:         var b: usize = 0; // button
306:         var l: usize = 0; // label
307:         var f: usize = 0; // field
308:
309:         switch (Rpanel.keyForIndex(n)) {
310:             Jpanel.name => {
311:                 val = json.get("PANEL").index(p).get(@tagName(Rpanel.keyForIndex(n)));
312:                 if (T.err) break;
313:
314:                 if (val.ctrlPack(Ctype.string))
315:                     ENRG.items[p].name = try std.fmt.allocPrint(allocator, "{s}", .{val.x?.string})
316:                 else
317:                     @panic(try std.fmt.allocPrint(allocator,
318:                         "Json Panel err_Field :{s}\n", .{@tagName(Rpanel.keyForIndex(n))}));
319:             },
320:             Jpanel.posx => {
321:                 val = json.get("PANEL").index(p).get(@tagName(Rpanel.keyForIndex(n)));
322:
```

```
323:         if (val.ctrlPack(Ctype.integer))
324:             ENRG.items[p].posx = @intCast(val.x?.integer)
325:         else
326:             @panic(try std.fmt.allocPrint(allocator,
327:                 "Json err_Field :{s}\n", .{@tagName(Rpanel.keyForIndex(n))}));
328:     },
329:     Jpanel.posy => {
330:         val = json.get("PANEL").index(p).get(@tagName(Rpanel.keyForIndex(n)));
331:
332:         if (val.ctrlPack(Ctype.integer))
333:             ENRG.items[p].posy = @intCast(val.x?.integer)
334:         else
335:             @panic(try std.fmt.allocPrint(allocator,
336:                 "Json err_Field :{s}\n", .{@tagName(Rpanel.keyForIndex(n))}));
337:     },
338:     Jpanel.lines => {
339:         val = json.get("PANEL").index(p).get(@tagName(Rpanel.keyForIndex(n)));
340:
341:         if (val.ctrlPack(Ctype.integer))
342:             ENRG.items[p].lines = @intCast(val.x?.integer)
343:         else
344:             @panic(try std.fmt.allocPrint(allocator,
345:                 "Json err_Field :{s}\n", .{@tagName(Rpanel.keyForIndex(n))}));
346:     },
347:     Jpanel.cols => {
348:         val = json.get("PANEL").index(p).get(@tagName(Rpanel.keyForIndex(n)));
349:
350:         if (val.ctrlPack(Ctype.integer))
351:             ENRG.items[p].cols = @intCast(val.x?.integer)
352:         else
353:             @panic(try std.fmt.allocPrint(allocator,
354:                 "Json err_Field :{s}\n", .{@tagName(Rpanel.keyForIndex(n))}));
355:     },
356:     Jpanel.cadre => {
357:         val = json.get("PANEL").index(p).get(@tagName(Rpanel.keyForIndex(n)));
358:
359:         if (val.ctrlPack(Ctype.string)) {
360:             ENRG.items[p].cadre = strToEnum(dds.CADRE, val.x?.string);
361:         } else @panic(try std.fmt.allocPrint(allocator,
362:             "Json err_Field :{s}\n", .{@tagName(Rpanel.keyForIndex(n))}));
363:     },
364:     Jpanel.title => {
365:         val = json.get("PANEL").index(p).get(@tagName(Rpanel.keyForIndex(n)));
366:
367:         if (val.ctrlPack(Ctype.string))
368:             ENRG.items[p].title = try std.fmt.allocPrint(allocator, "{s}", .{val.x?.string})
```



```
369:         else
370:             @panic(try std.fmt.allocPrint(allocator,
371:                 "Json err_Field :{s}\n", .{@tagName(Rpanel.keyForIndex(n))}));
372:     },
373:     //=====
374:     // BUTTON
375:     //=====
376:     Jpanel.button => {
377:         val = json.get("PANEL").index(p).get(@tagName(Rpanel.keyForIndex(n)));
378:         if (T.err) break;
379:
380:         var bt: DEFBUTTON = undefined;
381:         y = val.x?.array.items.len;
382:         z = 0;
383:         b = 0;
384:
385:         while (z < y) : (z += 1) {
386:             v = 0;
387:             while (v < Rbutton.count) : (v += 1) {
388:                 val = json.get("PANEL").index(p).get("button").index(b)
389:                     .get(@tagName(Rbutton.keyForIndex(v))
390:                 );
391:
392:                 switch (Rbutton.keyForIndex(v)) {
393:                     Jbutton.key => {
394:                         if (val.ctrlPack(Ctype.string)) {
395:                             bt.key = strToEnum(kbd, val.x?.string);
396:                         } else @panic(try std.fmt.allocPrint(allocator,
397:                             "Json err_Field :{s}.{s}\n", .{
398:                                 @tagName(Rpanel.keyForIndex(n)), @tagName(Rbutton.keyForIndex(v))
399:                             }));
400:                     },
401:                     Jbutton.show => {
402:                         if (val.ctrlPack(Ctype.bool))
403:                             bt.show = val.x?.bool
404:                         else
405:                             @panic(try std.fmt.allocPrint(allocator,
406:                                 "Json err_Field :{s}.{s}\n", .{
407:                                     @tagName(Rpanel.keyForIndex(n)), @tagName(Rbutton.keyForIndex(v))
408:                                 }));
409:                     },
410:                     Jbutton.check => {
411:                         if (val.ctrlPack(Ctype.bool))
412:                             bt.check = val.x?.bool
413:                         else
414:                             @panic(try std.fmt.allocPrint(allocator,
```



```
461:         @tagName(Rpanel.keyForIndex(n)), @tagName(Rlabel.keyForIndex(v))
462:     ));
463: },
464: Jlabel.posx => {
465:     if (val.ctrlPack(Ctype.integer)) {
466:         lb.posx = @intCast(val.x?.integer);
467:     } else @panic(try std.fmt.allocPrint(allocator,
468:         "Json err_Field :{s}.{s}\n", .{
469:             @tagName(Rpanel.keyForIndex(n)), @tagName(Rbutton.keyForIndex(v))
470:         }));
471: },
472: Jlabel.posy => {
473:     if (val.ctrlPack(Ctype.integer)) {
474:         lb.posy = @intCast(val.x?.integer);
475:     } else @panic(try std.fmt.allocPrint(allocator,
476:         "Json err_Field :{s}.{s}\n", .{
477:             @tagName(Rpanel.keyForIndex(n)), @tagName(Rbutton.keyForIndex(v))
478:         }));
479: },
480: Jlabel.text => {
481:     if (val.ctrlPack(Ctype.string))
482:         lb.text = try std.fmt.allocPrint(allocator, "{s}", .{val.x?.string})
483:     else
484:         @panic(try std.fmt.allocPrint(allocator,
485:             "Json err_Field :{s}.{s}\n", .{
486:                 @tagName(Rpanel.keyForIndex(n)), @tagName(Rlabel.keyForIndex(v))
487:             }));
488: },
489: Jlabel.title => {
490:     if (val.ctrlPack(Ctype.bool))
491:         lb.title = val.x?.bool
492:     else
493:         @panic(try std.fmt.allocPrint(allocator,
494:             "Json err_Field :{s}.{s}\n", .{
495:                 @tagName(Rpanel.keyForIndex(n)), @tagName(Rbutton.keyForIndex(v))
496:             }));
497:
498:     ENRG.items[p].label.append(lb) catch unreachable;
499: },
500:     }
501: }
502:
503:     l += 1;
504: }
505: },
506: //=====
```

```
507:      // FIELD
508:      //=====
509:
510:      Jpanel.field => {
511:          val = json.get("PANEL").index(p).get(@tagName(Rpanel.keyForIndex(n)));
512:          if ( T.err ) break ;
513:
514:          var sreftyp:[]const u8 = undefined;
515:
516:          var lf: DEFFIELD = undefined;
517:          y = val.x?.array.items.len;
518:          if( y == 0) break;
519:
520:          z = 0;
521:          f = 0;
522:          while (z < y) : (z += 1) {
523:              v = 0;
524:              while (v < Rfield.count) : (v += 1) {
525:
526:                  val = json.get("PANEL").index(p).get("field").index(f)
527:                      .get(@tagName(Rfield.keyForIndex(v))
528:                          );
529:
530:                  switch (Rfield.keyForIndex(v)) {
531:                      Jfield.name => { if (val.ctrlPack(Ctype.string))
532:                          lf.name = try std.fmt.allocPrint(allocator, "{s}", .{val.x?.string})
533:                      else
534:                          @panic(try std.fmt.allocPrint(allocator,
535:                              "Json err_Field :{s}.{s}\n", .{
536:                                  @tagName(Rpanel.keyForIndex(n)), @tagName(Rfield.keyForIndex(v))
537:                              }));
538:                  },
539:
540:                  Jfield.posx => { if (val.ctrlPack(Ctype.integer)) {
541:                      lf.posx = @intCast(val.x?.integer);
542:                  } else @panic(try std.fmt.allocPrint(allocator,
543:                      "Json err_Field :{s}.{s}\n", .{
544:                          @tagName(Rpanel.keyForIndex(n)), @tagName(Rbutton.keyForIndex(v))
545:                      }));
546:                  },
547:
548:                  Jfield.posy => { if (val.ctrlPack(Ctype.integer)) {
549:                      lf.posy = @intCast(val.x?.integer);
550:                  } else @panic(try std.fmt.allocPrint(allocator,
551:                      "Json err_Field :{s}.{s}\n", .{
552:                          @tagName(Rpanel.keyForIndex(n)), @tagName(Rbutton.keyForIndex(v))
```

```
553:         }}};
554:     },
555:
556:     Jfield.reftyp => {
557:         if (val.ctrlPack(Ctype.string)) {
558:             sreftyp = try std.fmt.allocPrint(allocator, "{s}", .{val.x?.string});
559:
560:         } else @panic(try std.fmt.allocPrint(allocator,
561:             "Json err_Field :{s}.{s}\n", .{
562:                 @tagName(Rpanel.keyForIndex(n)), @tagName(Rfield.keyForIndex(v))
563:             }));
564:
565:         lf.reftyp = strToEnum(dds.REFTYP ,sreftyp);
566:     },
567:
568:     Jfield.width => {if (val.ctrlPack(Ctype.integer)) {
569:         lf.width= @intCast(val.x?.integer);
570:     } else @panic(try std.fmt.allocPrint(allocator,
571:         "Json err_Field :{s}.{s}\n", .{
572:             @tagName(Rpanel.keyForIndex(n)), @tagName(Rbutton.keyForIndex(v))
573:         }));
574:     },
575:
576:     Jfield.scal => {if (val.ctrlPack(Ctype.integer)) {
577:         lf.scal= @intCast(val.x?.integer);
578:     } else @panic(try std.fmt.allocPrint(allocator,
579:         "Json err_Field :{s}.{s}\n", .{
580:             @tagName(Rpanel.keyForIndex(n)), @tagName(Rbutton.keyForIndex(v))
581:         }));
582:     },
583:
584:
585:     Jfield.text=>{
586:         lf.text="";
587:     },
588:
589:     Jfield.zwitch => {
590:         lf.zwitch= false;
591:     },
592:
593:     Jfield.requier => {
594:         if (val.ctrlPack(Ctype.bool)) {
595:             lf.requier = val.x?.bool ;
596:         } else @panic(try std.fmt.allocPrint(allocator,
597:             "Json err_Field :{s}.{s}\n", .{
598:                 @tagName(Rpanel.keyForIndex(n)), @tagName(Rbutton.keyForIndex(v))
```

```
599:         }}};
600:     },
601:
602:     Jfield.protect=> {
603:         if (val.ctrlPack(CType.bool)) {
604:             lf.protect= val.x?.bool ;
605:         } else @panic(try std.fmt.allocPrint(allocator,
606:             "Json err_Field :{s}.{s}\n", .{
607:                 @tagName(Rpanel.keyForIndex(n)), @tagName(Rbutton.keyForIndex(v))
608:             }));
609:     },
610:
611:     Jfield.edtcar => { if (val.ctrlPack(CType.string)) {
612:         lf.edtcar= try std.fmt.allocPrint(allocator, "{s}", .{val.x?.string});
613:     } else @panic(try std.fmt.allocPrint(allocator,
614:         "Json err_Field :{s}.{s}\n", .{
615:             @tagName(Rpanel.keyForIndex(n)), @tagName(Rfield.keyForIndex(v))
616:         }));
617:     },
618:
619:     Jfield.errmsg=> { if (val.ctrlPack(CType.string)) {
620:         lf.errmsg= try std.fmt.allocPrint(allocator, "{s}", .{val.x?.string});
621:     } else @panic(try std.fmt.allocPrint(allocator,
622:         "Json err_Field :{s}.{s}\n", .{
623:             @tagName(Rpanel.keyForIndex(n)), @tagName(Rfield.keyForIndex(v))
624:         }));
625:     },
626:
627:     Jfield.help=> { if (val.ctrlPack(CType.string)) {
628:         lf.help= try std.fmt.allocPrint(allocator, "{s}", .{val.x?.string});
629:     } else @panic(try std.fmt.allocPrint(allocator,
630:         "Json err_Field :{s}.{s}\n", .{
631:             @tagName(Rpanel.keyForIndex(n)), @tagName(Rfield.keyForIndex(v))
632:         }));
633:     },
634:
635:     Jfield.procfunc=> { if (val.ctrlPack(CType.string)) {
636:         lf.procfunc= try std.fmt.allocPrint(allocator, "{s}", .{val.x?.string});
637:     } else @panic(try std.fmt.allocPrint(allocator,
638:         "Json err_Field :{s}.{s}\n", .{
639:             @tagName(Rpanel.keyForIndex(n)), @tagName(Rfield.keyForIndex(v))
640:         }));
641:     },
642:
643:     Jfield.proctask => { if (val.ctrlPack(CType.string)) {
644:         lf.proctask = try std.fmt.allocPrint(allocator, "{s}", .{val.x?.string});
```

```
645:         } else @panic(try std.fmt.allocPrint(allocator,
646:         "Json err_Field :{s}.{s}\n", .{
647:         @tagName(Rpanel.keyForIndex(n)), @tagName(Rfield.keyForIndex(v))
648:         ));
649:         ENRG.items[p].field.append(lf) catch unreachable;
650:     },
651: }
652: }
653:
654:     f += 1;
655: }
656: },
657:
658: }
659: }
660: }
661: }
662:
663: //.....//
664: // Main function
665: //.....//
666: pub fn RstJson(XPANEL: *std.ArrayList(pnl.PANEL)) !void {
667:
668:     var my_file = try std.fs.cwd().openFile("fileJson.txt", .{});
669:     defer my_file.close();
670:
671:
672:     const file_size = try my_file.getEndPos();
673:     var buffer : []u8= allocator.alloc(u8, file_size) catch unreachable ;
674:
675:
676:     _= try my_file.read(buffer[0..buffer.len]);
677:
678:     jsonDecode(buffer) catch return;
679:
680:     deb_Log("zmodlRJson.txt");
681:
682:     plog(.schema).debug("\n{s}\n", .{buffer});
683:     plog(.DEBUG).debug("\nRead Json", .{});
684:
685:     for (ENRG.items) |pnlx| {
686:         plog(.Panel).debug("\n", .{});
687:         plog(.Panel).debug("{s}", .{pnlx.name});
688:         plog(.Panel).debug("{d}", .{pnlx.posx});
689:         plog(.Panel).debug("{d}", .{pnlx.posy});
690:         plog(.Panel).debug("{s}", .{pnlx.cadre});
```

```
691:     plog(.Panel).debug("{s}\n", .{pnlx.title});
692:
693:     plog(.Button).debug("\n", .{});
694:     for (pnlx.button.items) |r| {
695:         plog(.Button).debug("{s}", .{@tagName(r.key)});
696:         plog(.Button).debug("{any}", .{r.key});
697:         plog(.Button).debug("{", .{r.show});
698:         plog(.Button).debug("{", .{r.check});
699:         plog(.Button).debug("{s}\n", .{r.title});
700:     }
701:
702:     plog(.Label).debug("\n", .{});
703:     for (pnlx.label.items) |r| {
704:         plog(.Label).debug("{s}", .{r.name});
705:         plog(.Label).debug("{d}", .{r.posx});
706:         plog(.Label).debug("{d}", .{r.posy});
707:         plog(.Label).debug("{s}", .{r.text});
708:         plog(.Label).debug("{}\n", .{r.title});
709:     }
710:
711:     plog(.Field).debug("\n", .{});
712:     for (pnlx.field.items) |r| {
713:         plog(.Field).debug("{s}", .{r.name});
714:         plog(.Field).debug("{d}", .{r.posx});
715:         plog(.Field).debug("{d}", .{r.posy});
716:         plog(.Field).debug("{s}", .{@tagName(r.reftyp)});
717:         plog(.Field).debug("\n{d}", .{r.width});
718:         plog(.Field).debug("{d}", .{r.scal});
719:         plog(.Field).debug("\n{}", .{r.requier});
720:         plog(.Field).debug("{", .{r.protect});
721:         plog(.Field).debug("{s}\n", .{r.edtcar});
722:         plog(.Field).debug("{s}\n", .{r.errmsg});
723:         plog(.Field).debug("{s}\n", .{r.help});
724:         plog(.Field).debug("{s}\n", .{r.procfunc});
725:         plog(.Field).debug("{s}\n", .{r.proctask});
726:     }
727: }
728:
729: plog(.end).debug("End.\n", .{});
730:
731: end_Log();
732:
733:
734:
735: XPANEL.clearAndFree();
736:
```



```
737:     for (ENRG.items, 0..) |pnlx,idx| {
738:         var vPanel: pnl.PANEL= undefined;
739:         vPanel= pnl.initPanel(
740:             ENRG.items[idx].name,
741:             ENRG.items[idx].posx,
742:             ENRG.items[idx].posy,
743:             ENRG.items[idx].lines,
744:             ENRG.items[idx].cols,
745:             ENRG.items[idx].cadre,
746:             ENRG.items[idx].title);
747:
748:
749:
750:         for (pnlx.button.items) |p| {
751:             var vButton: btn.BUTTON= undefined;
752:
753:             vButton = btn.newButton(p.key,p.show,p.check,p.title);
754:
755:             vPanel.button.append(vButton)
756:             catch |err| { @panic(@errorName(err)); };
757:         }
758:
759:
760:         for (pnlx.label.items) |p| {
761:             var vLabel: lbl.LABEL= undefined;
762:
763:             if (p.title) vLabel = lbl.newTitle(p.name,p.posx,p.posy,p.text)
764:             else vLabel = lbl.newLabel(p.name,p.posx,p.posy,p.text);
765:
766:             vPanel.label.append(vLabel)
767:             catch |err| { @panic(@errorName(err)); };
768:         }
769:
770:
771:
772:         for (pnlx.field.items) |p| {
773:             var vField: fld.FIELD= undefined;
774:             switch(p.reftyp) {
775:
776:                 dds.REFTYP.TEXT_FREE => {
777:                     vField = fld.newFieldTextFree(
778:                         p.name,
779:                         p.posx,
780:                         p.posy,
781:                         p.width,
782:                         p.text,
```

```
783:         p.requier,  
784:         p.errmsg,  
785:         p.help,  
786:         p.regex,  
787:     );  
788:     vField.proctask= p.proctask;  
789:     vField.protect= p.protect;  
790: },  
791:  
792: dds.REFTYP.TEXT_FULL => {  
793:     vField = fld.newFieldTextFull(  
794:         p.name,  
795:         p.posx,  
796:         p.posy,  
797:         p.width,  
798:         p.text,  
799:         p.requier,  
800:         p.errmsg,  
801:         p.help,  
802:         p.regex,  
803:     );  
804:     vField.proctask= p.proctask;  
805:     vField.protect= p.protect;  
806: },  
807:  
808: dds.REFTYP.ALPHA => {  
809:     vField = fld.newFieldAlpha(  
810:         p.name,  
811:         p.posx,  
812:         p.posy,  
813:         p.width,  
814:         p.text,  
815:         p.requier,  
816:         p.errmsg,  
817:         p.help,  
818:         p.regex,  
819:     );  
820:     vField.proctask= p.proctask;  
821:     vField.protect= p.protect;  
822: },  
823:  
824: dds.REFTYP.ALPHA_UPPER => {  
825:     vField = fld.newFieldAlphaUpper(  
826:         p.name,  
827:         p.posx,  
828:         p.posy,
```

```
829:         p.width,
830:         p.text,
831:         p.requier,
832:         p.errmsg,
833:         p.help,
834:         p.regex,
835:     );
836:     vField.proctask= p.proctask;
837:     vField.protect= p.protect;
838: },
839:
840: dds.REFTYP.ALPHA_NUMERIC => {
841:     vField = fld.newFieldAlphaNumeric(
842:         p.name,
843:         p.posx,
844:         p.posy,
845:         p.width,
846:         p.text,
847:         p.requier,
848:         p.errmsg,
849:         p.help,
850:         p.regex,
851:     );
852:     vField.proctask= p.proctask;
853:     vField.protect= p.protect;
854: },
855:
856: dds.REFTYP.ALPHA_NUMERIC_UPPER => {
857:     vField = fld.newFieldAlphaNumericUpper(
858:         p.name,
859:         p.posx,
860:         p.posy,
861:         p.width,
862:         p.text,
863:         p.requier,
864:         p.errmsg,
865:         p.help,
866:         p.regex,
867:     );
868:     vField.proctask= p.proctask;
869:     vField.protect= p.protect;
870: },
871:
872: dds.REFTYP.PASSWORD => {
873:     vField = fld.newFieldPassword(
874:         p.name,
```

```
875:         p.posx,
876:         p.posy,
877:         p.width,
878:         p.text,
879:         p.requier,
880:         p.errmsg,
881:         p.help,
882:         p.regex,
883:     );
884:     vField.proctask= p.proctask;
885:     vField.protect= p.protect;
886: },
887:
888: dds.REFTYP.YES_NO => {
889:     vField = fld.newFieldYesNo (
890:         p.name,
891:         p.posx,
892:         p.posy,
893:         p.text,
894:         p.requier,
895:         p.errmsg,
896:         p.help,
897:     );
898:     vField.proctask= p.proctask;
899:     vField.protect= p.protect;
900:
901: },
902:
903: dds.REFTYP.SWITCH => {
904:     vField = fld.newFieldSwitch (
905:         p.name,
906:         p.posx,
907:         p.posy,
908:         p.zwitch,
909:         p.errmsg,
910:         p.help,
911:     );
912:     vField.proctask= p.proctask;
913:     vField.protect= p.protect;
914: },
915:
916: dds.REFTYP.DATE_FR => {
917:     vField = fld.newFieldDateFR (
918:         p.name,
919:         p.posx,
920:         p.posy,
```

```
921:         p.text,
922:         p.requier,
923:         p.errmsg,
924:         p.help,
925:     );
926:     vField.proctask= p.proctask;
927:     vField.protect= p.protect;
928: },
929:
930: dds.REFTYP.DATE_US => {
931:     vField = fld.newFieldDateUS (
932:         p.name,
933:         p.posx,
934:         p.posy,
935:         p.text,
936:         p.requier,
937:         p.errmsg,
938:         p.help,
939:     );
940:     vField.proctask= p.proctask;
941:     vField.protect= p.protect;
942: },
943:
944: dds.REFTYP.DATE_ISO => {
945:     vField = fld.newFieldDateISO (
946:         p.name,
947:         p.posx,
948:         p.posy,
949:         p.text,
950:         p.requier,
951:         p.errmsg,
952:         p.help,
953:     );
954:     vField.proctask= p.proctask;
955:     vField.protect= p.protect;
956: },
957:
958: dds.REFTYP.MAIL_ISO => {
959:     vField = fld.newFieldMail (
960:         p.name,
961:         p.posx,
962:         p.posy,
963:         p.width,
964:         p.text,
965:         p.requier,
966:         p.errmsg,
```

```
967:         p.help,
968:     );
969:     vField.proctask= p.proctask;
970:     vField.protect= p.protect;
971: },
972:
973: dds.REFTYP.TELEPHONE => {
974:     vField = fld.newFieldTelephone (
975:         p.name,
976:         p.posx,
977:         p.posy,
978:         p.width,
979:         p.text,
980:         p.requier,
981:         p.errmsg,
982:         p.help,
983:         p.regex,
984:     );
985:     vField.proctask= p.proctask;
986:     vField.protect= p.protect;
987: },
988:
989: dds.REFTYP.DIGIT => {
990:     vField = fld.newFieldDigit (
991:         p.name,
992:         p.posx,
993:         p.posy,
994:         p.width,
995:         p.text,
996:         p.requier,
997:         p.errmsg,
998:         p.help,
999:         p.regex,
1000:     );
1001:     vField.proctask= p.proctask;
1002:     vField.protect= p.protect;
1003:     vField.edtcar= p.edtcar;
1004: },
1005:
1006: dds.REFTYP.UDIGIT => {
1007:     vField = fld.newFieldUDigit (
1008:         p.name,
1009:         p.posx,
1010:         p.posy,
1011:         p.width,
1012:         p.text,
```

```
1013:         p.requier,
1014:         p.errmsg,
1015:         p.help,
1016:         p.regex,
1017:     );
1018:     vField.proctask= p.proctask;
1019:     vField.protect= p.protect;
1020:     vField.edtcar= p.edtcar;
1021: },
1022:
1023: dds.REFTYP.DECIMAL => {
1024:     vField = fld.newFieldDecimal (
1025:         p.name,
1026:         p.posx,
1027:         p.posy,
1028:         p.width,
1029:         p.scal,
1030:         p.text,
1031:         p.requier,
1032:         p.errmsg,
1033:         p.help,
1034:         p.regex,
1035:     );
1036:     vField.proctask= p.proctask;
1037:     vField.protect= p.protect;
1038:     vField.edtcar= p.edtcar;
1039: },
1040:
1041: dds.REFTYP.UDECIMAL => {
1042:     vField = fld.newFieldUDecimal (
1043:         p.name,
1044:         p.posx,
1045:         p.posy,
1046:         p.width,
1047:         p.scal,
1048:         p.text,
1049:         p.requier,
1050:         p.errmsg,
1051:         p.help,
1052:         p.regex,
1053:     );
1054:     vField.proctask= p.proctask;
1055:     vField.protect= p.protect;
1056:     vField.edtcar= p.edtcar;
1057: },
1058:
```

```
1059:         dds.REFTYP.FUNC => {
1060:             vField = fld.newFieldFunc (
1061:                 p.name,
1062:                 p.posx,
1063:                 p.posy,
1064:                 p.width,
1065:                 p.text,
1066:                 p.requier,
1067:                 p.procfunc,
1068:                 p.errmsg,
1069:                 p.help,
1070:             );
1071:             vField.proctask= p.proctask;
1072:             vField.protect= p.protect;
1073:         },
1074:         else => {},
1075:     }
1076:
1077:     vPanel.field.append(vField)
1078:         catch |err| { @panic(@errorName(err)); };
1079: }
1080:
1081: XPANEL.append(vPanel) catch unreachable;
1082: }
1083:
1084:
1085: ENRG.clearAndFree();
1086:
1087: dds.deinitUtils();
1088:
1089: }
```