```
PROCEDURE Begin*;
  VAR self: Tasks.Context;
    i: INTEGER;
    stage: Htc.StageT;
    req: Htc.Request;
    head, body: Buffers.Kit;
    tid: LONGINT;
BEGIN
  Tasks.Begin(tid);
  self := Tasks.I();
  head.New(Buffers.dir, {});
  body.New(Buffers.dir, {});
  req := Htc.dir.NewRequest();
  req.Open(Ht.GET, Ht.http, "forum.drakon.su", "/viewtopic.php?f=153&t=3489")
    .SendReceive(NIL, head.wr, body.wr);
  self.Await(req, Htc.completed, NIL).Await(req, Htc.failed, NIL).TimeoutSec(15).Pass;
  IF self.CauseObj() = NIL THEN
    Log.String("Timeout!"); Log.Ln;
    req.Cancel
```

```
VAR
  NumTyp-: RECORD
    char-, integer-, real-, int64-, real32-, real64-: INTEGER
  END;

  Sym-: RECORD
    null-, times-, slash-, div-, mod-,
    and-, plus-, minus-, or-, eql-,
    neq-, lss-, leq-, gtr-, geq-,
    in-, is-, arrow-, dollar-, period-,
    comma-, colon-, upto-, rparen-, rbrak-,
    rbrace-, of-, then-, do-, to-,
    by-, not-,
    lparen-, lbrak-, lbrace-, becomes-,
    number-, nil-, string-, ident-, semicolon-,
    bar-, end-, else-, elsif-, until-,
    if-, case-, while-, repeat-, for-,
    loop-, with-, exit-, return-, array-,
    record-, pointer-, begin-, const-, type-,
    var-, out-, procedure-, close-, import-,
    module-, eof: INTEGER
  END;

BEGIN
  Enum.InitN(Sym);
  Enum.InitN(NumTyp)
END Op2eScanner.
```

```
MODULE FncAuth;
 IMPORT ...

 VAR
  user-: Mk.TypeT;
   (* Q.id *)
   email-: Mk.StringT;
   active-: Mk.BoolT;
   regtime-, lastVisit-: Mk.LongintT;

  userReferer-: Mk.TypeT;
   (* user => user *)

  password-: Mk.TypeT;
   (* Q.id *)
   hash-: Mk.StringT;
   salt-: Mk.StringT;
```

```
PROCEDURE Init;
BEGIN
 Tok.InitTokens(Fnc.projId, handler);
 Tok.InitFullName(user);
 Tok.InitFullName(password);
 user.Attr(Q.id).Attr(email).Attr(active)
  .Attr(regtime).Attr(lastVisit).EndAttrs;
 password.Attr(Q.id).Attr(hash)
  .Attr(salt).EndAttrs
 ...
END Init;
```

```
TYPE
 Token = POINTER TO EXTENSIBLE RECORD (KrlObject.Object)
  id-: LONGINT;
  mod-, var-, name-: POINTER TO ARRAY OF CHAR;
  props-: SET;
  base-: ARRAY 12 OF KrlTokens.Token;
  ext-, next-, ext_0-: KrlTokens.Token;
  level-: INTEGER;
  handler-: KrlTokens.Handler;
  opts-: ARRAY 12 OF KrlTokens.Token;
  facets-, defaultVal-: KrlTokens.Token;
 END;
```

```
TYPE
 Token = POINTER TO EXTENSIBLE RECORD (KrlObject.Object)
  id-: LONGINT;
  mod-, var-, name-: POINTER TO ARRAY OF CHAR;
  props-: SET;
  base-: ARRAY 12 OF KrlTokens.Token;
  ext-, next-, ext_0-: KrlTokens.Token;
  level-: INTEGER;
  handler-: KrlTokens.Handler;
  opts-: ARRAY 12 OF KrlTokens.Token;
  facets-, defaultVal-: KrlTokens.Token;
 END;
```

```
OptionT* = POINTER TO EXTENSIBLE RECORD (EntityT) END;
TroubleT* = POINTER TO EXTENSIBLE RECORD (EntityT) END;
ErrorT* = POINTER TO EXTENSIBLE RECORD (TroubleT) END;
WarningT* = POINTER TO EXTENSIBLE RECORD (TroubleT) END;

TypeT* = POINTER TO EXTENSIBLE RECORD (Token) END;

RecTypeT* = POINTER TO EXTENSIBLE RECORD (TypeT)
  type-: KrlMeta.Type;
  tmp: POINTER TO ARRAY OF ANYPTR
END;

AnyIntT* = POINTER TO EXTENSIBLE RECORD (FieldT)
  min-, max-: Mem.HugestInt
END;

ByteT* = POINTER TO EXTENSIBLE RECORD (AnyIntT) END;
SIntT* = POINTER TO EXTENSIBLE RECORD (AnyIntT) END;
IntT* = POINTER TO EXTENSIBLE RECORD (AnyIntT) END;
LIntT* = POINTER TO EXTENSIBLE RECORD (AnyIntT) END;
```

```
PROCEDURE HandleMsg* (obj: ANYPTR; VAR par1, par2: ANYREC;
  IN par3: ANYREC);
  VAR t: Mk.Token;
BEGIN
  WITH par3: SqlOpts.TokenPar DO
    t := par3.t
  ELSE
  END;
  WITH par1: SqlOpts.ColOpts DO
    (* user *)
    IF t = email THEN
      par1.size := 127;
      par1.columnOpts := {SqlOpts.colUnique}
    (* password *)
    ELSIF t = hash THEN
      par1.size := 128 (* для SHA-512 *)
    ELSIF t = salt THEN
      par1.size := 16
    END
  ELSE
  END
END HandleMsg;
```

```
PROCEDURE CreateTabs*;
BEGIN
  KrfdSql.CreateTab(user, '', TRUE, '', '');
  KrfdSql.CreateTab(password, '', TRUE, '', '');
  KrfdSql.CreateTab(userReferer, '', TRUE, '', '')
END CreateTabs;

PROCEDURE DropTabs*;
BEGIN
  KrfdSql.DropTab(password);
  KrfdSql.DropTab(user);
  KrfdSql.DropTab(userReferer)
END DropTabs;
```

```
PROCEDURE CreatePerson* (c: Brokers.Controller; id: LONGINT; OUT ok: BOOLEAN);
  VAR req: Mk.DocData; errors: SET;
BEGIN
  Mk.GetTmp(req);
  req.doc.Writer().El(MSocial.updatePerson)
    .El(MSocial.person)
      .LInt(Q.id, id)
      .Str(MSocial.name, "").Str(MSocial.surname, "")
      .Str(MSocial.patronymic, "").Str(MSocial.company, "")
      .Str(MSocial.about, "")
    .End(MSocial.person)
  .End(MSocial.updatePerson).Z;
  Brokers.DoSimp(c, req, {}, errors);
  ok := errors = {}
END CreatePerson;
```

```
PROCEDURE InitUserReferals;
BEGIN
  userReferals.decl := FncDb.qdir.NewDeclarations("MOD")
    .List(Q.L1).List(usr).Int(fromT).Int(toT);

  userReferals.stat := FncDb.qdir.NewStatements(userReferals.decl)
    .Select(Q.L1).From(Auth.user)
      .LinkedTo(Auth.userReferer, usr)
      .Between(Auth.regtime, fromT, toT)
    .Do();

  userReferals.fetch := FncDb.qdir.NewFetch(userReferals.decl);
  userReferals.fetch.Struct()
    .Elc(Auth.user, Q.fromList).Str(Q.list, Q.L1)
        .El1(MSocial.person)
        .Elc(Auth.userReferer, Q.outRel)
          .El(Auth.user)
            .El1(MSocial.person)
          .End(Auth.user)
        .End(Auth.userReferer)
    .End(Auth.user).Z
END InitUserReferals;
```

```
TYPE
  ErrorT* = POINTER TO EXTENSIBLE RECORD (Lib.ErrorT) END;
  WarningT* = POINTER TO EXTENSIBLE RECORD (Lib.WarningT) END;

  TwinErr* = POINTER TO EXTENSIBLE RECORD (ErrorT) END;
  TwinPosSigErr* = POINTER TO EXTENSIBLE RECORD (TwinErr) END;

  OverflowErr* = POINTER TO EXTENSIBLE RECORD (ErrorT) END;
  BeltSkewErr* = POINTER TO EXTENSIBLE RECORD (ErrorT) END;

  NoRouteErr* = POINTER TO EXTENSIBLE RECORD (ErrorT) END;

VAR
  errorT-: ErrorT;
  warningT-: WarningT;
  twinErr-: TwinErr;
  twinPosSigErr-: TwinPosSigErr;
  overflowErr-: OverflowErr;
  beltSkewErr-: BeltSkewErr;
  noRouteErr-: NoRouteErr;
```

```
VAR errors: ARRAY 32 OF Tok.Token;

Компилятор зануляет, т.к. это указатели.

Tok.Incl(errors, beltError);

IF Tok.HasSame(errors, Tok.warningT)
THEN

Log.String(errors[i].name);

binLog.LInt(errors[i].id);

tok := Tok.ById(id)
```

```
TYPE
  Value* = RECORD
    field-: Toks.FieldT;
    ptr: S.PTR;
    data-: ValueData
  END;

  ValueData* = ARRAY 16 OF BYTE;

  Record* = ARRAY OF Value;
```

```
TYPE
  Value = RECORD
    field-: KrlTokens.FieldT;
    data-: KrlVariants.ValueData;
    (IN v: Value) IsArray (): BOOLEAN, NEW;
    (IN v: Value) Len (): INTEGER, NEW;
    (IN v: Value) Bool (): BOOLEAN, NEW;
    (IN v: Value) Bools (): POINTER TO ARRAY OF BOOLEAN, NEW;
    (IN v: Value) Byte (): BYTE, NEW;
    (IN v: Value) Bytes (): POINTER TO ARRAY OF BYTE, NEW;
    (IN v: Value) Ints (): POINTER TO ARRAY OF INTEGER, NEW;
    (IN v: Value) Obj (): KrlProps.Object, NEW;
    (IN v: Value) Ptr (): ANYPTR, NEW;
    (IN v: Value) Ptrs (): POINTER TO ARRAY OF ANYPTR, NEW;
    ....
  END
```

```
DEFINITION KrlJournaling;

  IMPORT KrlLife, KrlObject, KrlVariants;

  TYPE
    Writer = POINTER TO ABSTRACT RECORD (KrlLife.Object)
        Flush (hard: BOOLEAN), NEW, ABSTRACT;
        Transect, NEW, ABSTRACT;
        Write (rec: KrlVariants.Record), NEW, ABSTRACT
    END;

END KrlJournaling.
```

```
PROCEDURE WriteSwitchOnEv*
  (event: SwitchOnEv; node: Tok.Token);
    VAR rec: ARRAY 16 OF Var.Value;
BEGIN
    rec[0].PutTok(Var.recType, event);
    rec[1].PutLInt(ErcJrn.timestampSec, KrlDates.UnixT());
    rec[2].PutTok(ErcJrn.nodeSign, node);
    wr.Write(rec)
END WriteSwitchOnEv;
```

```
List* = POINTER TO EXTENSIBLE RECORD (Props.Object)
  next*, prev*,
  firstSub*, lastSub*, super*: List;
  rec*: POINTER TO Record;
  names*: POINTER TO ARRAY OF POINTER TO ARRAY OF CHAR;
  class*: Toks.Token;
  ext*: ANYPTR
END;
```

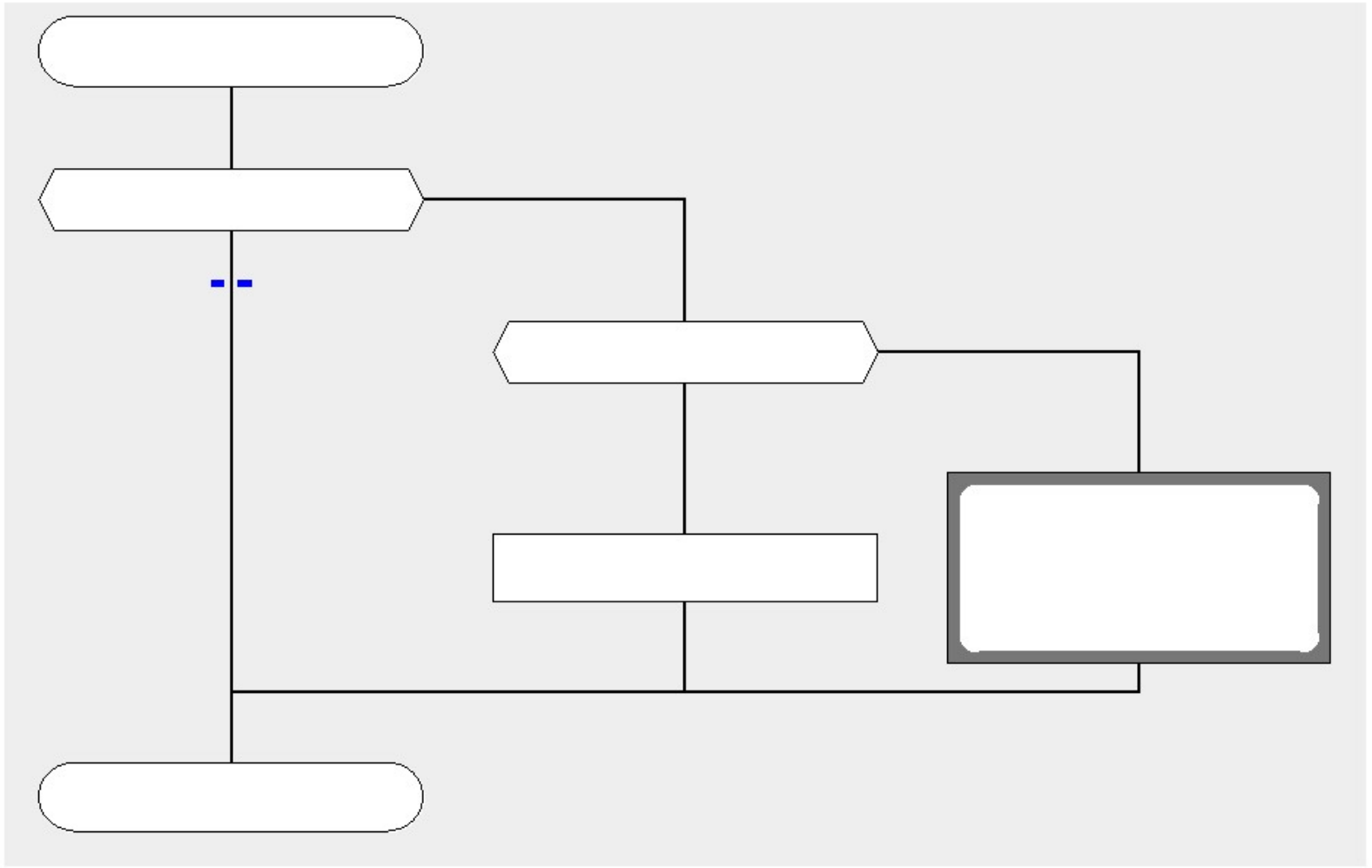
```
PROCEDURE SwOn_EstimateModeSig (obj: Props.Object);
  VAR contSig, contCmd: BOOLEAN;
BEGIN
  contCmd := Tags.Bool(obj, Lib.out, Pwr.contCmd);
  contSig := Tags.Bool(obj, Lib.in, Pwr.contSig);
  IF contCmd & ~contSig THEN
    Tags.SetTok(obj, NIL, Lib.modeSig, Lib.isActive_Starting)
  ELSIF ~contCmd & contSig THEN
    Tags.SetTok(obj, NIL, Lib.modeSig, Lib.isActive_Stopping)
  ELSIF contCmd THEN
    Tags.SetTok(obj, NIL, Lib.modeSig, Lib.isActive)
  ELSE (* оборудование выключено, но проверяем команды *)
    IF Tags.Tok(obj, Lib.own, Lib.modeCmd) IS Lib.Off_0 THEN
      Tags.SetTok(obj, NIL, Lib.modeSig, Lib.isInactive_0)
    ELSE
      Tags.SetTok(obj, NIL, Lib.modeSig, Lib.isActive_Starting)
    END
  END
END SwOn_EstimateModeSig;
```

```
 PROCEDURE Tran_Init* (obj: Props.Object; type, base: Tok.Token; context:
ANYPTR; VAR par: ANYREC);
 BEGIN
  obj.AddNew(Lib.in).AddNew(Lib.out)
    .AddMethod(Control.controlDown, Tran_ControlDown, NIL)
    .AddMethod(Control.controlUp, Tran_ControlUp, NIL).Z;
  obj.Obj(Lib.in)
    .AddNew(Pwr.contSig)
    .AddNew(Dry.moveSig)
    .AddNew(Dry.overflowErrSig).Z;
  obj.Obj(Lib.out)
    .AddNew(Pwr.contCmd).Z
 END Tran_Init;
```

```
VAR
  myTab-: Db.Table; (*! BEGIN *)
    col1-: Tok.StrT; (*! VARCHAR, N = 5, M = 10 *)
    (*! SUB anotherColumn, AnMod.anCol2 *)
  (*! END *)
```

```
VAR
 array: ARRAY 128 OF INTEGER;
 (*! DO = KrlNotes.LoadArray,
  VALUES = (12, 14, 17, 18) *)
```

```
VAR
  myAlgo-: Drakon.Algo;
  (*!
```



```
  *)
```

```
OP2E
   USE DB := RdbDB;

MODULE AisAuthController;
   IMPORT Auth := AisAuth, Pers := AisPers;

   PROCEDURE HandleAuth (….);
   BEGIN
     DB.EXEC ON dbConn
       SELECT FROM Auth.user, Pers.person
         WHERE Pers.year > fromYear;
   END HandleAuth;

END AisAuthController.
```

```
OP2E
  USE XS := XscCompiler;
  DO
    XS.param := ....;
    XS.Install;

MODULE MyModule;
  ......
```

```
OP2E USE L := Op2eLib;
MODULE MyMod;
  IMPORT ....;

    text := L.TEXT
      ..,dwkdpow
      dwdwi
      dwpdw
    END;
```

```
VAR
  State-: RECORD
     s1-, s2-, s3-: INTEGER
  END;
  L.INIT
     myOpts.opt1 := 135;
  END;

VAR
  arr: ARRAY 2, 10 OF INTEGER;
L.INIT
  L.SET_ARRAY arr TO (1, 2, 3, 4, 5),
    (1, 2, 3, 4, 7);
END;
```

```
OP2E USE X := Op2eExt;
MODULE MyModule;

    X.SECTION (* MyClass *)

      CONST ...

      TYPE ...

      PROCEDURE ...

    END_SECTION;

END MyModule;
```

```
f.Open;
X.DEFER f.Close; ... END;
```

```
X.LET y := abc + def;
Proc(a, b, c, X.LET(s, 255), X.LET x);
```

```
GetAsJson(req, ..., X.TMP json);
ConnectDb(..., X.LET conn);
X.DEFER conn.Close END;
MapToDb(conn, pattern, ..., json, X.LET res);
```

```
X.FUN F (x, y: INTEGER): INTEGER DEF x + y;
```

```
X.SWTO AnotherProc(x, y, z);
END Proc;
```