ChnEB

JEDNOTKA DEFINUJÍCÍ KOMUNIKAČNÍ PROTOKOL E-BISYNC

Příručka uživatele a programátora

***SofCon*®spol. s r.o.**
Střešovická 49
162 00 Praha 6
tel/fax: +420 220 180 454
E-mail: sofcon@sofcon.cz
www: http://www.sofcon.cz

Verze dokumentu 1.20

Informace v tomto dokumentu byly pečlivě zkontrolovány a SofCon věří, že jsou spolehlivé, přesto SofCon nenese odpovědnost za případné nepřesnosti nebo nesprávnosti zde uvedených informací.
SofCon negarantuje bezchybnost tohoto dokumentu ani programového vybavení, které je v tomto dokumentu popsané. Uživatel přebírá informace z tohoto dokumentu a odpovídající programové vybavení ve stavu, jak byly vytvořeny a sám je povinen provést validaci bezchybnosti produktu, který s použitím zde popsaného programového vybavení vytvořil.
SofCon si vyhrazuje právo změny obsahu tohoto dokumentu bez předchozího oznámení a nenese žádnou odpovědnost za důsledky, které z toho mohou vyplynout pro uživatele.

Datum vydání: 05.04.2004
Datum posledního uložení dokumentu: 05.04.2004
(Datum vydání a posledního uložení dokumentu musí být stejné)

Obsah :

1. O dokumentu

1.1. Revize dokumentu

Verze dokumentu	Verze SW	Autor	Datum vydání	Popis změn
1.00	1.XX	Wi		První vydání.
1.10	4.XX	Tu	22.05.2003	Úprava dokumentu dle ISO9000.
1.20	4.XX	Wil	05.04.2004	Přidány konstanty res_ErrFrame, res_ErrLen.

1.2. Účel dokumentu

Tento dokument slouží jako popis jednotky definující komunikační protokol E—YNC.

1.3. Rozsah platnosti

Určen pro programátory a uživatele programového vybavení SofCon.

1.4. Související dokumenty

Pro čtení tohoto dokumentu je potřeba seznámit se s manuálem „ChnVirt“ popisujícím základní rodičovský prvek pro tvorbu komunikačních objektů.

Pro zabezpečení dat protokolu se používá algoritmu definovaném v knihovně „XOR8“.

Popis formátu verze knihovny a souvisejících funkcí je popsán v manuálu „LibVer“.

2. Termíny a definice

Používané termíny a definice jsou popsány v samostatném dokumentu „Termíny a definice“.

3. Úvod

Knihovna ChnEB definuje komunikační objekt **tChnEB**, který je dědicem od rodičovského komunikačního objektu tChnVirt. Instance objektu tChnEB representuje vyšší komunikační vrstvu v komunikačním kanálu. Transformuje předávaná data mezi komunikačními objekty nižších vrstev, které provádějí fyzický přenos, a aplikací nebo případně další vyšší komunikační vrstvou. Objekt tChnEB definuje formát protokolu používaného při komunikaci se zařízeními EUROTHERM a obstarává zabezpečení dat, vkládání a vyjímání nadbytečností, tak jak to tento protokol předepisuje. Fyzický přenos dat je zajištěn prostřednictvím nižší komunikační vrstvy. Určení, přes jakou fyzickou komunikační vrstvu bude komunikace probíhat, je voleno až parametry nastavovací metody ChSetParam.

Knihovna rovněž definuje objekt **tAddChnEB**, který je dědicem od rodičovského objektu tAddChnVirt. Objekt tAddChnEB zajistí, aby daný komunikační objekt (objekt tChnEB) byl k aplikaci připojen a popřípadě zajistí vytvoření instance tohoto objektu. Po přilinkování této jednotky do aplikace (příkazem "uses ChnEB"), se jméno objektu tChnEB automaticky vloží do seznamu správců komunikačních objektů pro případné použití.

Protože je objekt **tChnEB** dědicem rodičovského komunikačního objektu **tChnVirt**, jsou v této příručce popsány jen odlišnosti a speciality pro tento druh sériové komunikace. Ostatní naleznete v příručce **ChnVirt**. Některé použité konstanty a typy jsou předdefinované v jednotce **ChnTypes**.

4. Konstanty a typy

```
cVerNo = např. $0251; { BCD formát }
cVer   = např. '02.51,07.08.2003';
```

Číslo verze jednotky v BCD tvaru a v textové podobě včetně datumu změny.

```
cName = 'EB';
```

Konstanta **cName** definuje jméno komunikačního objektu **tChnEB**.

```
tMessType = (tpRead, tpWrite, tpACK, tpNAK, tpBS);
```

Výčtový typ **tMessType** definuje několik základních typů zpráv.

Typ **tpRead** je pro zprávu pro čtení dat od Master stanice nebo pro odpověď na tuto zprávu od Slave stanice.

Typ **tpWrite** je pro zprávu pro zápis dat od Master stanice.

Typ **tpACK** je pro krátké pozitivní potvrzení na zápis dat od Slave stanice nebo pro požadavek na čtení dalšího parametru od Master stanice po úspěšném čtení libovolného parametru.

Typ **tpNAK** je pro krátké negativní potvrzení na zápis dat od Slave stanice nebo pro požadavek na čtení téhož parametru od Master stanice po úspěšném čtení libovolného parametru.

Typ **tpBS** je pro požadavek na čtení předešlého parametru od Master stanice po úspěšném čtení libovolného parametru.

```
tParam = (tpWrongCode, tpFloat, tpHexa);
```

Výčtový typ **tParam** definuje několik základních datových formátů.

Typ **tpWrongCode** nedefinuje žádný datový formát, znamená pouze, že Slave stanice nerozpoznala parametr zadaný mnemonickým kódem.

Typ **tpFloat** definuje formát maximální délky 6 bytů pro reálná a celá čísla a to jak kladná tak i záporná v jejich ASCII representaci (čísla '0' až '9', desetinná tečka '.', a znaménko '-' popřípadě i '+').
Typ **tpHexa** definuje formát pevné délky 5 bytů pro celá kladná čísla v hexadecimální podobě v jejich ASCII representaci (čísla a všechny znaky anglické abecedy) a úvodním znakem '>', který odlišuje tento formát od všech ostatních datových formátů.

4.1. Konstanty řídících znaků protokolu

```
STX   = $02;
ETX   = $03;
EOT   = $04;
ENQ   = $05;
ACK   = $06;
NAK   = $0F;
BS    = $08;
```

Konstanty **STX**, **ETX**, **EOT**, **ENQ**, **ACK**, **NAK** a **BS** definují kódy kontrolních a zabezpečovacích znaků protokolu.

4.2. Konstanty výsledků přijímacího automatu

V této kapitole jsou popsány definice chybových kódů, které může vracet metoda **ChReceiveResult** v průběhu přijímacího automatu.

```
res_ErrFrame = $20;
```

Výsledek **res_ErrFrame** indikuje chybu rámce přijaté zprávy.

```
res_ErrSum   = $21;
```

Výsledek **res_ErrSum** indikuje chybu kontrolního součtu při přijmu zprávy.

```
res_ErrLen   = $22;
```

Výsledek **res_ErrLen** indikuje chybu délky zprávy.

```
res_ErrVal   = $23;
```

Výsledek **res_ErrVal** indikuje chybu při převodu čísla na textový řetězec a naopak.

4.3. Struktury přijímacích a vysílacích bufferů

```
pSendRecord = ^tSendRecord;
tSendRecord = record
  case MessType      : tMessType of
    tpRead,
    tpWrite :
      (Code          : String[2];
       case Par      : tParam of
         tpWrongCode :
           ();
         tpFloat     :
           (Float    : Real;);
         tpHexa      :
           (Dummy1_1 : array[1..1]of byte;
            Hex      : Word;);
      );
    tpACK ,
    tpNAK ,
    tpBS  :
      ( );
```

```
  end;
```

TSendRecord je typ variantního záznamu, který svou strukturou odpovídá datům protokolu E-Bysinc pro zasílání zpráv, přičemž je zbaven nadbytečností, které jsou při vysílání doplněny. Položka **MessType** udává typ zprávy (čtení, zápis dat apod.). Při čtení a zápisu dat se používají položky **Code** a **Par** pro stanici Master i Slave. Položka **Code** obsahuje mnemonický kód čteného či zapisovaného parametru v podobě textového vyjádření číslic a znaků anglické abecedy, přičemž se rozlišují malá a velká písmena. Položka **Par** udává datovou strukturu čteného či zapisovaného parametru (Slave stanice může také tuto položku místo datové struktury parametru nastavit na typ udávající špatně zadaný mnemonický kód - viz. výčtový typ tParam). Master stanice používá navíc při zápisu dat položku **Hex** (číslo v hexadecimálním formátu) nebo **Float** (číslo ve formátu celého či reálného čísla a to jak kladné tak i záporné), kdežto Slave stanice používá tyto položky naopak při čtení dat. Položka **Dummy1_1** nemá žádný zvláštní význam, v záznamu je definována pouze pro zarovnání položky Hex na adresu dělitelnou dvěma.

```
tRecRecord  = tSendRecord;
pRecRecord  = ^tRecRecord;
```

TRecRecord je typ variantního záznamu, který svou strukturou odpovídá datům protokolu E-Bysinc pro příjem zpráv, přičemž je zbaven nadbytečností, které jsou při příjmu odstraněny. Svou vnitřní strukturou je shodný s typem **TSendRecord**.

5. Objekty

5.1. tChnEB

5.1.1. Položky

```
CH_RTick   : Boolean;
```

Položka **CH_RTick** označuje, že je vykonávaná činnost přijímacího automatu. Tato položka se používá pro ladění.

```
CH_SBuff   : Pointer;
```

Položka **CH_SBuff** definuje ukazatel na vysílací buffer.

```
CH_MSBuff  : Word;
```

Položka **CH_MSBuff** definuje délku vysílacího bufferu.

```
CH_LRMess  : Word;
```

Položka **CH_LRMess** definuje délku přijímané zprávy.

```
CH_RSum    : tXor8;
```

Položka **CH_RSum** se používá pro počítání kontrolního součtu přijímače.

```
CH_SSum    : tXor8;
```

Položka **CH_SSum** se používá pro počítání kontrolního součtu vysílače.

```
CH_Master  : Boolean;
```

Položka **CH_Master** definuje, je-li stanice zapojena v síti jako nadřazená jednotka (Master) nebo jako podřízená jednotka (Slave).

5.1.2. Metody

5.1.2.1. Init konstruktor

```
constructor Init;
```

Konstruktor **Init** slouží k vytvoření a inicializaci instance komunikačního objektu. Ve svém těle je nejprve zavolána zděděná metoda **Init** (inherited Init) od rodičovského objektu tChnVirt a poté jsou inicializovány položky objektu. Tělo konstruktoru vypadá následovně:

```
inherited Init;
CH_Type    := cName;
CH_Name    := CH_Type;
CH_NumName := ChNumName(CH_Type);
CH_RTick   := false;
CH_SBuff   := nil;
CH_MSBuff  := 0;
CH_LRMess  := 0;
CH_Master  := true;
CH_RDNodeS := '00';
CH_FlRecSTX:= false;
CH_FlAdrOk := false;
CH_ss      := '';
```

5.1.2.2. ChInitParam konstruktor

```
constructor ChInitParam(const S: tParamStr);
```

Konstruktor **ChInitParam** slouží ke zkrácenému vytvoření instance komunikačního objektu s definovaným nastavením parametrů kanálu. Ve svém těle nejprve volá konstruktor **Init** a poté metodu **ChSetParam**.

5.1.2.3. Done destruktor

```
destructor Done;
```

Destruktor **Done** slouží ke zrušení instance komunikačního objektu. Pokud je alokován vysílací buffer, je odstraněn z paměti. Na konci destruktoru je volána zděděná metoda **Done** od přímého rodičovského objektu pro uzavření podřízené komunikační vrstvy.

5.1.2.4. ChSetOneParam procedura

```
function ChSetOneParam(const S: tWordString; var CmdL: tCmd)
         : tChResult;
```

Metoda **ChSetOneParam** slouží k dekódování a nastavení jednoho konkrétního parametru, který je zadán v parametru S. Tato metoda se volá v aplikaci prostřednictvím metody **ChSetParam**. Metoda **ChSetOneParam** komunikačního objektu tChnEB dekóduje tyto parametry:

MAS=MASTER / SLAVE

Parametrem **MAS** ("Master or Slave") se určuje, zda je jednotka v komunikační síti jako Master (nadřízená) nebo jako Slave (podřízená).

LSB=Size

Parametrem **LSB** ("Length of Send Buffer") je alokován nový vysílací buffer **CH_MSBuff** dané velikosti Size.

NOD=Node

Parametrem **NOD** ("Node") se určuje číslo (adresa) stanice **CH_Node** v komunikační síti. Node může nabývat hodnot <0..254>. Adresa 255 je určena protokolem implicitně pro všechny stanice společně.

DNO=DNode

Parametrem **DNO** ("Destination Node") se určuje číslo (adresa) stanice **CH_DNode** v komunikační síti, které budou zprávy určeny. Tuto položku je možno také definovat prostřednictvím metody **ChDestNode**. DNode může nabývat hodnot <0..254>. Adresa 255 je určena protokolem implicitně pro všechny stanice společně.

5.1.2.5. ChGetParam funkce

```
function  ChGetParam(const S: TParamStr): TParamStr;
```

Metoda **ChGetParam** navrací nastavené hodnoty parametrů komunikačního objektu. Nejprve vrátí nastavení parametrů rodičovského komunikačního objektu tChnVirt a poté k nim připojí seznam svých parametrů. Seznam parametrů je uveden výše u popisu metody **ChSetOneParam**.

5.1.2.6. ChConnect procedura

```
procedure ChConnect;
```

Metoda **ChConnect** zavolá zděděnou metodu **ChConnect** od přímého rodičovského objektu a pokud nenastala žádná chyba, nastaví automat přijímače **CH_RCtrl** do počátečního stavu pro příjem zprávy v protokolu E-Bysinc.

5.1.2.7. ChDisConnect procedura

```
procedure ChDisConnect;
```

Metoda **ChDisConnect** zavolá zděděnou metodu **ChDisConnect** od přímého rodičovského objektu a pokud nenastala žádná chyba, nastaví automat přijímače **CH_RCtrl** do neaktivního stavu, aby se nepřijímaly žádné zprávy.

5.1.2.8. ChSend procedura

```
procedure ChSend(Buff : Pointer; Len : Word);
```

Metoda **ChSend** způsobí započetí vysílání zprávy podle protokolu E-Bysinc na podkladu záznamu typu tSendRecord, na který ukazuje parametr Buff. Parametr Len udává délku vysílacího bufferu pro vysílání. Tento parametr není nutno správně naplnit skutečnou délkou zprávy, jelikož protokol umí tuto délku získat automaticky podle vyplněného záznamu vysílacího bufferu. Před voláním této metody se musí záznam správně naplnit daty (viz níže).

5.1.2.9. ChReceiveReady funkce

```
function  ChReceiveReady: tChState;
```

Metoda **ChReceiveReady** způsobí provedení kroku přijímacího automatu na základě volání metody **ChReceiveTick**. Jako svoji funkční hodnotu vrací aktuální stav automatu přijímače komunikačního kanálu, který je uložen v položce **CH_RCtrl**. Zpravidla se provádí test pouze na stabilní stav **CHS_ReceiveReady** (který znamená, že byla přijata nějaká zpráva), protože ostatní stavy jsou stavy probíhajícího příjmu.

5.1.2.10. ChReceive procedura

```
procedure ChReceive(var Len: Word);
```

Metoda **ChReceive** provede přijmutí celé zprávy a její uložení do přijímacího bufferu, který svou vnitřní strukturou odpovídá datům záznamu typu tRecRecord a byl definován metodou **ChReceiveBuffer**. Metoda naplní buffer pouze patřičnými položkami typu tRecRecord, úvodní a zakončovací řídicí znaky a kontrolní součet ze zprávy metoda vyhodnotí a pro uživatele odstraní. Odpověď na zprávu, ať došla v pořádku nebo porušená, generuje uživatel sám pomocí metody **ChSend**.

5.1.2.11. ChReceiveFlush procedura

```
procedure ChReceiveFlush;
```

Metoda **ChReceiveFlush** způsobí vyprázdnění přijímacích bufferů a nastavení stavu automatu přijímače na počátek příjmu zpráv v protokolu E-Bysinc.

5.1.2.12. ChGetNode procedura

```
procedure ChGetNode(var SNode, DNode : tNode);
```

Po volání metody **ChGetNode** je do proměnné **SNode** uloženo číslo (adresa) stanice, která zprávu odeslala, a do proměnné **DNode** číslo (adresa) stanice, pro kterou byla zpráva určena. Tuto metodu má smysl volat po přijetí zprávy metodou **ChReceive**.

5.1.2.13. ChReceiveTick procedura

```
procedure ChReceiveTick;
```

Metoda **ChReceiveTick** způsobí provedení jednoho či více kroků automatu přijímače. Je nutné ji periodicky volat při přijímání. Metoda **ChReceiveTick** je rovněž automaticky volána v metodě **ChReceiveReady**.

5.2. tAddChnEB

Typ **tAddChnEB** je typem objektu, který slouží k definování prvku v seznamu správců komunikačních objektů (tzv. správce komunikačního objektu tChnEB v seznamu správců). Objekt tAddChnEB je dědicem od rodičovského objektu **tAddChnVirt**.

5.2.1. Metody

5.2.1.1. ChInit funkce

```
function ChInit: pChnVirt;
```

Metoda **ChInit** slouží k vytvoření instance komunikačního objektu tChnEB a ukazatel na instanci tohoto objektu vrací jako svoji funkční hodnotu.

6. Popis protokolu E-BISYNC

Tento protokol, udávaný též pod názvem SIC800, používá firma Eurotherm u regulátorů řady 800 a 900. Uvedený protokol je popsán v normě ANSI-X3.28-2.5-A4. Pro přenos jednotlivých znaků po sériové lince se používá toto nastavení: 7 datových bitů, sudá parita a 1 stopbit.

6.1. Struktura zpráv a odpovědí pro čtení dat

Master stanice posílá požadavek na čtení dat následujícího formátu:

EOT	GID	GID	UID	UID	CODE	ENQ

Nastavení jednotlivých položek ve struktuře vysílacího bufferu:

```
MessType := tpRead;
Code     := ... např. 'PV';
délka zprávy = 4
```

Pokud **Slave** stanice rozeznala CODE posílá zpět požadovaná data.

STX	CODE	DATA...	ETX	BCC

Nastavení jednotlivých položek ve struktuře vysílacího bufferu:

```
MessType := tpRead;
Code     := stejný jako při vysílání Master stanicí např. 'PV';
Par      := tpHexa nebo tpFloat;
```

pokud Par = tpHexa:

```
Hex      := ... např. $0123;
délka zprávy = 8
```

pokud Par = tpFloat:

```
Float    := ... např. -10.58
délka zprávy = 11
```

Pokud **Slave** stanice nerozpoznala CODE posílá zpět zprávu následujícího formátu:

STX	CODE	EOT

Nastavení jednotlivých položek ve struktuře vysílacího bufferu:

```
MessType := tpRead;
Code     := stejný jako při vysílání Master stanicí např. 'PV';
Par      := tpWrongCode;
délka zprávy = 5
```

Pokud **Master** stanice přečetla úspěšně nějaká data může pro čtení následujících, předchozích nebo těch samých dat použít některou z těchto zkrácených zpráv na které je stejná odpověď od slave stanice jako v případě dlouhé čtecí zprávy.

ACK

- Čtení následujících dat. Tato zpráva má stejný význam jako kdyby se vyslala předchozí zpráva pro čtení dat, ale s CODE zvětšeným o 1.

Nastavení jednotlivých položek ve struktuře vysílacího bufferu:

```
MessType := tpACK;
délka zprávy = 1
```

NAK

- Čtení těch samých dat. Tato zpráva má stejný význam jako kdyby se vyslala předchozí zpráva pro čtení dat se stejným CODE.

Nastavení jednotlivých položek ve struktuře vysílacího bufferu:

```
MessType := tpNAK;
délka zprávy = 1
```

BS

- Čtení předchozích dat. Tato zpráva má stejný význam jako kdyby se vyslala předchozí zpráva pro čtení dat, ale s CODE zmenšeným o 1.

Nastavení jednotlivých položek ve struktuře vysílacího bufferu:

```
MessType := tpBS;
délka zprávy = 1
```

Význam položek:

EOT, **ENQ**, **STX**, **ETX**, **ACK**, **NAK** a **BS** - řídící znaky protokolu (viz. příslušné konstanty těchto znaků)

GID - identifikátor skupiny daného zařízení = CH_DNode div 10 + Ord ('0')

UID - identifikátor jednotky daného zařízení = CH_DNode mod 10 + Ord ('0')

CODE- mnemonický kód daného parametru - 2 ASCII číslice nebo znaky anglické abecedy (malá a velká písmena se rozlišují)

DATA- datové pole zprávy podle datového formátu parametru (viz. výčtový typ tParam)

BCC - kontrolní (bytový) součet zprávy = CODE xor DATA xor ETX

6.2. Struktura zpráv a odpovědí pro zápis dat

Master stanice posílá zprávu pro zápis dat následujícího formátu:

EOT	GID	GID	UID	UID	STX	CODE	DATA...	ETX	BCC

Nastavení jednotlivých položek ve struktuře vysílacího bufferu:

```
MessType := tpWrite;
Code     := ... např. 'PV';
Par      := tpFloat nebo tpHexa
```

pokud Par = tpHexa:

```
Hex      := ... např. $0123;
délka zprávy = 8
```

pokud Par = tpFloat:

```
Float    := ... např. -10.58
délka zprávy = 11
```

Slave stanice odpovídá některou z následujících zpráv:

ACK

- Zápis dat proběhl úspěšně.

Nastavení jednotlivých položek ve struktuře vysílacího bufferu:

```
MessType := tpACK;
délka zprávy = 1
```

NAK

- Zpráva byla přijata, ale zápis nelze provést.

Nastavení jednotlivých položek ve struktuře vysílacího bufferu:

```
MessType := tpNAK;
délka zprávy = 1
```

7. Příklad

Následující příklad ukazuje způsob vyslání zprávy pro zápis dat ve formátu tpHexa s čekáním na odpověď. Fyzický přenos se realizuje prostřednictvím knihovny ChnCom.

```
uses
  uString,
  ChnVirt,
  ChnCom,
  ChnEB;

const
  ParamStr : tParamStr =
             'NAM=EB LSB=500 NOD=1 DNO=2 ' +
             'NAM=COM COM=1 IRQ=4 BD=9600 BIT=7 STOP=1 PAR=E '+
             'LRB=1000';
var
  Chn   : pChnVirt;
  SMess : pMaSendRecord;
  RMess : pMaRecRecord;
  LSMess: word;
  LRMess: word;

begin
  ...
  New(SMess);
  New(RMess);
  ...
  { vytvoření instance Chn }
  Chn:=ChnCollection^.ChNewInit(ChnTecom.cName);
  with Chn^ do
  begin
    { nastavení parametrů komunikace }
    ChSetParam(ParamStr);
    if ChResult<>res_Ok then WriteLn('Chyba');
    ChOpen;
    repeat
      if ChResult<>res_Ok then WriteLn('Chyba');
    until ChReady=CHS_Open;
    if ChResult<>res_Ok then WriteLn('Chyba');
    { definování místa, kam se má přjatá zpráva uložit }
    ChReceiveBuffer(RMess,SizeOf(RMess^));
    if ChReceiveResult<>res_Ok then WriteLn('Chyba');
    ChConnect;
    repeat
      if ChResult<>res_Ok then WriteLn('Chyba');
    until ChReady=CHS_Connect;
    if ChResult<>res_Ok then WriteLn('Chyba');
    ...
    { naplnění zprávy daty (např. zpráva Connect)}
    with SMess^ do
    begin
      MessType:=tpWrite;
      Code    :='PV';
      Par     :=tpHexa;
      Hex     :=$0102;
      LSMess  :=0; { pro tuto knihovnu není povinné }
    end;
    { vyslání zprávy }
```

```
    if ChSendReady=CHS_SendReady then
    begin
      ChSend(SMess, LSMess);
      { čekání na odvysílání zprávy }
      repeat
        if ChSendResult<>res_Ok then WriteLn('Chyba');
      until ChSendReady=CHS_SendReady;
      if ChSendResult<>res_Ok then WriteLn('Chyba');
      ...
    end;
    ...
    { čekání na příjem zprávy }
    while not ChReceiveReady=CHS_ReceiveReady do
    begin
      if ChReceiveResult<>res_Ok then WriteLn('Chyba');
    end;
    { příjem zprávy }
    ChReceive(LRMess);
    if ChReceiveResult<>res_Ok then WriteLn('Chyba')
    else
      { dekódování správné odpovědi (např. odpověď na Connect)}
      with RMess^ do
      begin
        if MessType = tpACK then
          writeln('Ok')
        else
          writeln('Chyba');
      end;
    ...
    { ukončení }
    ChDisConnect;
    repeat
      if ChResult<>res_Ok then WriteLn('Chyba');
    until ChReady=CHS_DisConnect;
    if ChResult<>res_Ok then WriteLn('Chyba');
    ChClose;
    repeat
      if ChResult<>res_Ok then WriteLn('Chyba');
    until ChReady=CHS_Close;
    if ChResult<>res_Ok then WriteLn('Chyba');
  end;
  { zrušení instance Chn }
  Dispose(Chn,Done);
  ...
end.
```