ChnFT1_2

JEDNOTKA DEFINUJÍCÍ KOMUNIKAČNÍ PROTOKOL DLE NORMY EN 60870-5-2 FORMÁT FT 1.2

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 1.10

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: 20.05.2003
Datum posledního uložení dokumentu: 20.05.2003
(Datum vydání a posledního uložení dokumentu musí být stejné)

Upozornění:
V dokumentu použité názvy výrobků, firem apod. mohou být ochrannými známkami nebo registrovanými ochrannými známkami příslušných vlastníků.

Obsah :

1.O dokumentu 5
1.1. Revize dokumentu 5
1.2. Účel dokumentu 5
1.3. Rozsah platnosti 5
1.4. Související dokumenty 5
2.Termíny a definice 5
3.Úvod 6
4.Popis konstant a typů 6
4.1. Řídící znaky protokolu 7
4.2. Kódy služeb protokolu 7
4.3. Konstanty výsledků přijímacího automatu 8
4.4. Struktury přijímacích a vysílacích bufferů 8
4.5. Velikosti hlaviček vysílacích a přijímacích bufferů 9
5.Objekty 10
5.1. tChnFT1_2 10
5.1.1. Položky 10
5.1.2. Metody 10
5.1.2.1. Init konstruktor 10
5.1.2.2. ChInitParam konstruktor 11
5.1.2.3. Done destruktor 11
5.1.2.4. ChSetOneParam funkce 11
5.1.2.5. ChGetParam funkce 12
5.1.2.6. ChConnect procedura 12
5.1.2.7. ChDisConnect procedura 12
5.1.2.8. ChSend procedura 12
5.1.2.9. ChReceiveReady funkce 12
5.1.2.10. ChReceive procedura 12
5.1.2.11. ChReceiveFlush procedura 13
5.1.2.12. ChGetNode procedura 13
5.1.2.13. ChReceiveTick procedura 13
5.2. tAddChnFT1_2 13
5.2.1. Metody 13
5.2.1.1. ChInit funkce 13
6.Struktura zpráv protokolu 13
6.1. Význam položek 14
6.1.1. Struktura řídicího pole CODE 15
7.Příklad 16

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	5.XX	Tu	20.05.2003	Úprava dokumentu dle ISO9000.

1.2. Účel dokumentu

Tento dokument slouží jako popis jednotky definující komunikační protokol dle normy **EN 60870-5-2** formát **FT 1.2**.

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 rozhraní svých potomků.

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 definuje komunikační protokol podle normy **ČSN EN 60870-5-2** shodné s normou **IEC 870-5-2** formátu přenosového rámce **FT 1.2** pro vyvážený i nevyvážený (Master-Slave) přenos. Obstarává zabezpečení dat, vkládání a vyjímání nadbytečností, tak jak to tento protokol předepisuje. Fyzický přenos dat je zajištěn prostřednictvím nižší komunikační vrstvy.

Při vyváženém přenosu je vysílací stanice označována jako *primární* a přijímací stanice jako *sekundární*. U nevyváženého přenosu je nadřízená stanice označována jako *Master* a podřízena jako *Slave*. Jelikož struktura zpráv vysílané primární i Master stanicí a zpráv vysílané sekundární i Slave stanici se takřka neliší, budeme dále v tomto manuálu rozumět pod pojmem Master stanice i stanici primární a pod pojmem Slave stanice i stanici sekundární. Na případné odlišnosti bude upozorněno.

Knihovna ChnFT1_2 definuje komunikační objekt **tChnFT1_2**, který je dědicem od rodičovského komunikačního objektu tChnVirt. Instance objektu tChnFT1_2 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. Určení, přes jakou fyzickou komunikační vrstvu bude komunikace probíhat, je voleno až parametry nastavovací metody ChSetParam.

Knihovna rovněž definuje objekt **tAddChnFT1_2**, který je dědicem od rodičovského objektu tAddChnVirt. Objekt tAddChnFT1_2 zajistí, aby daný komunikační objekt (objekt tChnFT1_2) byl k aplikaci připojen a popřípadě zajistí vytvoření instance tohoto objektu. Po přilinkování této jednotky do aplikace (příkazem "uses ChnFT1_2"), se jméno objektu tChnFT1_2 automaticky vloží do seznamu správců komunikačních objektů pro případné použití.

Protože je objekt **tChnFT1_2** 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. Popis konstant a typů

```
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 = 'FT1_2';
```

Konstanta **cName** definuje jméno komunikačního objektu **tChnFT1_2**.

```
MaxTBuf = 65500;
```

Maximální velikost vysílacího bufferu.

```
MaxDataLen  = 247;
```

Konstanta **MaxDataLen** definuje maximální délku přenášeného datového bloku v bytech (pole Data ve strukturách vysílacích a přijímacích bufferů).

```
tRec = (Znk, NoZnk, Err);
```

Typ pro vnitřní řídicí proměnnou automatu přijímače.

```
tMessType = (tpSACK, tpSNAK, tpShortM, tpLongM);
```

Výčtový typ **tMessType** definuje základní typy zpráv.
Typ **tpSACK** je pro krátké pozitivní potvrzení od Slave stanice na zápis dat.
Typ **tpSNAK** je pro krátké negativní potvrzení od Slave stanice na zápis dat.
Typ **tpShortM** je pro zprávu pevné délky datového pole od Master i Slave stanice.
Typ **tpLongM** je pro zprávu proměnné délky datového pole od Master i Slave stanice.

4.1. Řídící znaky protokolu

```
SD1   = $10;
SD2   = $68;
ED    = $16;
SACK  = $E5;
SNAK  = $A2;
```

Konstanta **SD1** definuje kód znaku pro začátek zprávy s pevnou délkou datového pole.
Konstanta **SD2** definuje kód znaku pro začátek zprávy s proměnnou délkou datového pole.
Konstanta **ED** definuje kód znaku pro konec zprávy s pevnou i proměnnou délkou datového pole.
Konstanta **SACK** definuje kód znaku pro krátké pozitivní potvrzení od Slave stanice na zápis dat.
Konstanta **SNAK** definuje kód znaku pro krátké negativní potvrzení od Slave stanice na zápis dat.

4.2. Kódy služeb protokolu

Tyto kódy služeb FC (Function Code) jsou definované výše zmíněnou normou, ale knihovna umí pracovat obecně se všemi číselnými kódy FC, tj. hodnoty 0 až 15.

Za každým níže uvedeným kódem je uvedeno, jestli daný kód služby je pro vyvážený „V“ nebo v nevyvážený „N“ přenos nebo pro oba.

Následující konstanty znaků určují kód služby (FC - Function Code) dané zprávy od primární/Master stanice:

```
FCM_1stSetTransmit=  0; {V i N}
```

Počáteční nastavení dálkového spojení. Položku FCV ve vysílacím bufferu je nutno nastavit na hodnotu False.

```
FCM_1stSetUserFun =  1; {V i N}
```

Počáteční nastavení provozu uživatele. Položku FCV ve vysílacím bufferu je nutno nastavit na hodnotu False

```
FCM_CheckLink     =  2; {jen V}
```

Zkušební funkce pro linku. Položku FCV ve vysílacím bufferu je nutno nastavit na hodnotu True.

```
FCM_UserDataReply =  3; {V i N}
```

Uživatelská data s čekáním na odpověď. Položku FCV ve vysílacím bufferu je nutno nastavit na hodnotu True.

```
FCM_UserDataNoRep =  4; {V i N}
```

Uživatelská data bez čekání na odpověď. Položku FCV ve vysílacím bufferu je nutno nastavit na hodnotu False.

```
FCM_ReqReplyAcces =  8; {jen N}
```
Očekávána odpověď určující požadavek přístupu. Položku FCV ve vysílacím bufferu je nutno nastavit na hodnotu False.
```
FCM_ReqLinkState  =  9; {V i N}
```
Dotaz na stav linky. Položku FCV ve vysílacím bufferu je nutno nastavit na hodnotu False.
```
FCM_ReqUserData1  = 10; {jen N}
```
Požadavek na uživatelská data třídy 1. Položku FCV ve vysílacím bufferu je nutno nastavit na hodnotu True.
```
FCM_ReqUserData2  = 11; {jen N}
```
Požadavek na uživatelská data třídy 2.Položku FCV ve vysílacím bufferu je nutno nastavit na hodnotu True.

Následující konstanty znaků určují kód služby (FC - Function Code) dané zprávy od sekundární/Slave stanice:

```
FCS_Ack           =  0; {V i N}
```
ACK: pozitivní potvrzení.
```
FCS_NackBusy      =  1; {V i N}
```
NACK: Zpráva nebyla přijata, linka je obsazena.
```
FCS_UserData      =  8; {jen N}
```
Uživatelská data.
```
FCS_DataNotAcces  =  9; {jen N}
```
NACK: Požadovaná data nejsou přístupná.
```
FCS_LinkSAccesReq = 11; {V i N}
```
Stav linky nebo požadavek přístupu pro nevyvážený přenos.

4.3. Konstanty výsledků přijímacího automatu

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

```
res_ErrSum   = $20;
```
Výsledek **res_ErrSum** indikuje chybu kontrolního součtu při příjmu zprávy.
```
res_ErrLen   = $21;
```
Výsledek **res_ErrLen** indikuje chybnou délku bloku dat.
```
res_ErrCode  = $22;
```
Výsledek **res_ErrCode** indikuje chybu v řídicím poli zprávy.
```
res_ErrFrame = $23;
```
Výsledek **res_ErrFrame** indikuje chybu rámce zprávy.

4.4. Struktury přijímacích a vysílacích bufferů

```
tData = array[1..MaxDataLen]of byte;
```
Typ pole vlastních dat datové zprávy.

```
pSendRecord = ^tSendRecord;
tSendRecord = record
  MessType : tMessType;
  FCB_ACD  : boolean;
  FCV_DFC  : boolean;
  FC       : byte;
  Data     : tData;
end;
```

TSendRecord je typ variantního záznamu, který svou strukturou odpovídá datům protokolu posílaným Master/primární stanicí i Slave/sekundární stanicí, přičemž je zbaven nadbytečností, které jsou při vysílání doplněny.

Položka **MessType** určuje typ datové zprávy tpShortM nebo tpLongM (hodnoty tpSACK a tpSNAK jsou pro Master/primární stanici nepřípustné).

Položka **FCB_ACD** pro Master/primární stanici určuje hodnotu bitu načítání rámců – střídající se bit po sobě jdoucích služeb SEND/CONFIRM nebo REQUEST/RESPOND. Bit načítání rámců je použit pro eliminování ztrát a duplicit přenosu informací.

Položka **FCB_ACD** je pro Slave/sekundární stanici využívána jen při nevyváženém přenosu pro určení požadavek přístupu. Je-li FCB_ACD=True, je požadován přístup pro přenos dat třídy 1 (data s vyšší prioritou). V opačném případě (FCB_ACD=False) je požadován přenos dat třídy 2 (data s normální – nižší prioritou).

Položka **FCV_DFC** pro Master/primární stanici určuje platnost bitu načítání rámců FCB_ACD. FCV_DFC=False znamená ignorování bitu načítání rámců. Pokud je FCV_DFC=False měla by být i FCB_ACD=False.

Položka **FCV_DFC** vysílaná Slave/sekundární stanicí informuje Master/primární stanici o možnosti příjmu dalších dat. Je-li FCV_DFC=True, znamená to, že další zprávy od Master/primární stanice mohou způsobit přeplnění daty. V opačném případě (FCV_DFC=False) jsou další zprávy od Master/primární stanice přípustné.

Položka **FC** (Function Code) určuje kód požadované služby a může nabývat hodnot 0 až 15.

Pole **Data** je obecné bytové pole pro další využití aplikace.

Vyhodnocování platného bitu FCB_ACD a případné zopakování zprávy s nezměněným FCB_ACD musí provést sama aplikace či případná vyšší komunikační (linková) vrstva.

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

TRecRecord je typ variantního záznamu, který svou strukturou odpovídá datům protokolu přijímaným Master/primární stanicí i Slave/sekundární stanicí, přičemž je zbaven nadbytečností, které jsou při příjmu odstraněny. Svou vnitřní strukturou je shodný s typem **TSendRecord**.

4.5. Velikosti hlaviček vysílacích a přijímacích bufferů

```
cSizeHeadSndMsg = 1+1+1+1;
      { SizeOf(MessType) + SizeOf(FCB_ACD) + SizeOf(FCV_DFC) +
      SizeOf(FC)}
```

Konstanta **cSizeHeadSndMsg** určuje velikost hlavičky vysílacího bufferu Master i Slave stanice bez pole Data.

```
cSizeHeadRecMsg = cSizeHeadSndMsg;
```

Konstanta **cSizeHeadRecMsg** určuje velikost hlavičky přijímacího bufferu Master i Slave stanice bez pole Data.

5. Objekty

5.1. tChnFT1_2

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 : tSum8;`

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

`CH_SSum : tSum8;`

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

`CH_MaSl : Boolean;`

Položka **CH_MaSl** určuje použití nevyváženého (Master-Slave) přenosu nebo vyváženého (primární-sekundární) přenosu.

`CH_Master : Boolean;`

Položka **CH_Master** definuje, je-li v případě nevyváženého přenosu stanice zapojena v síti jako nadřazená jednotka (Master) nebo jako podřízená jednotka (Slave), nebo v případě vyváženého přenosu je stanice v primární nebo sekundární roli.

`CH_LenShortM : Byte;`

Položka **CH_LenShortM** definuje délku pole Data (v bytech) zprávy s datovým polem pevné délky.

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 zavolá zděděný konstruktor **Init** (inherited Init) od rodičovského objektu tChnVirt a inicializuje 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_MaSl      := true;
CH_Master    := true;
CH_LenShortM := 4;
IData        := 0;
```

5.1.2.2. ChInitParam konstruktor

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

Konstruktor **ChInitParam** je sloučením konstruktoru **Init** a metody **ChSetParam**. Slouží ke zkrácenému vytvoření instance komunikačního objektu s nastavením parametrů komunikace.

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 funkce

```
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 tChnFT1_2 dekóduje tyto parametry:

MS=ON / OFF

Parametrem **MS** ("Master-Slave or Primary-Secondary") se určuje nevyvážený či vyvážený přenos. Hodnota ON znamená nevyvážený (Master-Slave) přenos.

MAS=MASTER / SLAVE

Parametrem **MAS** ("Master or Slave") se určuje, zda je jednotka v komunikační síti při nevyváženém přenosu jako Master (nadřízená) nebo jako Slave (podřízená). Tento parametr se používá, pokud parametr MS=ON.

PRI=ON / OFF

Parametrem **PRI** ("Primary or Secundary") se určuje, zda je jednotka v komunikační síti při vyváženém přenosu jako Primární (vysílající) nebo jako Sekundární (přijímající). Tento parametr se používá, pokud parametr MS=OFF.

LSB=Size

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

NOD=Node

Parametrem **NOD** ("Node") se určuje číslo (adresa) stanice **CH_Node** v komunikační síti. Hodnota Node může být v rozsahu 0 až 255.

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**. Hodnota DNode může být v rozsahu 0 až 255.

LEN=lll

Parametrem **LEN** ("Length of Data in tpShortM message") se určuje délka pole Data (v bytech) datové zprávy pevné délky (tpShortM).

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.

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 na podkladu záznamu typu tMaSendRecord nebo tSlSendRecord, na který ukazuje parametr Buff. Parametr Len udává délku vysílacího bufferu pro vysílání. Tento parametr je nutno správně zadat v případě datové zprávy (tpShortM nebo tpLongM) a to následovně: Len = počet vysílaných bytů v poli Data + cSizeHeadMaSndMsg pro Master stanici nebo Len = počet vysílaných bytů v poli Data + cSizeHeadSlSndMsg pro Slave stanici. V případě zpráv tpSACK a tpSNAK se tento parametr ignoruje. 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 tMaRecRecord nebo tSlRecRecord a byl definován metodou **ChReceiveBuffer**. Metoda naplní buffer pouze patřičnými položkami typu tMaRecRecord nebo tSlRecRecord, úvodní a zakončovací řídicí znaky a kontrolní součet ze zprávy metoda vyhodnotí a pro

uživatele odstraní. Parametr Len metoda naplní následovně: Při zprávě tpSACK nebo tpSNAK je Len = 1, při zprávě tpShortM nebo tpLongM je Len = počet přijatých bytů v poli Data + cSizeHeadRecMsg pro Master i Slave stanici.

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.

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. tAddChnFT1_2

Typ **tAddChnFT1_2** je typem objektu, který slouží k definování prvku v seznamu správců komunikačních objektů (tzv. správce komunikačního objektu tChnFT1_2 v seznamu správců). Objekt tAddChnFT1_2 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 tChnFT1_2 a ukazatel na instanci tohoto objektu vrací jako svoji funkční hodnotu.

6. Struktura zpráv protokolu

Zpráva s datovým polem pevné délky vysílaná z Master stanice do Slave stanice.

SD1	CODE	DNO	DATA	SUM	ED

Zpráva s datovým polem pevné délky vysílaná ze Slave stanice do Master stanice.

SD1	CODE	SNO	DATA	SUM	ED

Zpráva s datovým polem proměnné délky vysílaná z Master stanice do Slave stanice.

SD2	LEN	LEN	SD2	CODE	DNO	DATA	SUM	ED

Zpráva s datovým polem proměnné délky vysílaná ze Slave stanice do Master stanice.

SD2	LEN	LEN	SD2	CODE	SNO	DATA	SUM	ED

Krátké pozitivní potvrzení od stanice Slave.

SACK

Krátké negativní potvrzení od stanice Slave.

SNAK

6.1. Význam položek

SD1 - úvodní znak pro datovou zprávu pevné délky (start delimiter 1)
pevná hodnota $10

SD2 - úvodní znak pro datovou zprávu proměnné délky (start delimiter 2)
pevná hodnota $68

LEN - délka zprávy (počet bytů položek CODE+DNO(SNO)+DATA)

CODE - řídicí pole obsahující řídicí bity přenosu zpráv a Function Code FC (viz. dále)

DNO - adresa cílové stanice (destination address)
hodnota 0..255

SNO - adresa zdrojové stanice (source address)
hodnota 0..255

DATA - vlastní tělo datové zprávy
maximálně 247 bytů

SUM - kontrolní součet (frame check sum)
bytový součet všech bytů položek CODE, DNO(SNO) a DATA se zanedbáním vyšších řádů vzniklých přenosem.

ED - koncový znak (end delimiter)
pevná hodnota $16

SACK - krátké pozitivní potvrzení (short acknowledge)
pevná hodnota $E5

SNAK - krátké negativní potvrzení (short non-acknowledge)
pevná hodnota $A2

6.1.1. Struktura řídicího pole CODE

Struktura řídicího pole CODE pro Master stanici při nevyváženém přenosu:

7	6	5	4	3 2 1 0
RES	PRM	FCB	FCV	FC

RES – rezervováno (nepoužívá se)
PRM = 1 – zpráva od Master stanice
FCB – střídající se bit načítání rámců
FCV – platnost používání FCB (při FCV = 0 se FCB ignoruje)
FC – (Function Code) kód funkce zprávy

Struktura řídicího pole CODE pro Slave stanici při nevyváženém přenosu:

7	6	5	4	3 2 1 0
RES	PRM	ACD	DFC	FC

RES – rezervováno (nepoužívá se)
PRM = 0 – zpráva od Slave stanice
ACD – požadavek přístupu ACD = 0 – požadavek přístupu pro přenos dat třídy 2
ACD = 1 – požadavek přístupu pro přenos dat třídy 1
DFC – řízení toku dat DFC = 0 – jsou přípustné další zprávy
DFC = 1 – další zprávy mohou způsobit přeplnění daty
FC – (Function Code) kód funkce zprávy

Struktura řídicího pole CODE pro primární stanici při vyváženém přenosu:

7	6	5	4	3 2 1 0
DIR	PRM	FCB	FCV	FC

DIR – fyzický směr přenosu DIR = 0 – ze stanice A do stanice B
DIR = 1 – ze stanice B do stanice A
PRM = 1 – zpráva od primární stanice
FCB – střídající se bit načítání rámců
FCV – platnost používání FCB (při FCV = 0 se FCB ignoruje)
FC – (Function Code) kód funkce zprávy

Struktura řídicího pole CODE pro sekundární stanici při vyváženém přenosu:

7	6	5	4	3 2 1 0
DIR	PRM	ACD	DFC	FC

DIR – fyzický směr přenosu DIR = 0 – ze stanice A do stanice B
DIR = 1 – ze stanice B do stanice A
PRM = 0 – zpráva od sekundární stanice
RES – rezervováno (nepoužívá se)
DFC – řízení toku dat DFC = 0 – jsou přípustné další zprávy
DFC = 1 – další zprávy mohou způsobit přeplnění daty
FC – (Function Code) kód funkce zprávy

7. Příklad

Následující příklad ukazuje způsob vyslání datové zprávy proměnné délky ze stanice Master do stanice Slave s dekódováním přijaté odpovědi. Fyzický přenos se realizuje prostřednictvím knihovny ChnCom.

```
uses
  uString,
  ChnVirt,
  ChnCom,
  ChnFT1_2;

const
  ParamStr : tParamStr =
             'NAM=FT1_2 MS=ON MASTER=ON LSB=500 NOD=1 DNO=2 ' +
             'NAM=COM COM=1 IRQ=4 BD=9600 BIT=8 STOP=1 ' +
             'PAR=E LRB=1000';

var
  Chn   : pChnVirt;
  SMess : pSendRecord;
  RMess : pRecRecord;
  LSMess: word;
  LRMess: word;
  I     : byte;

begin
  ...
  New(SMess);
  New(RMess);
  ...
  { vytvoření instance Chn }
  Chn:=ChnCollection^.ChNewInit(ChnFT1_2.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');
    ...
    { příklad naplnění zprávy daty }
    with SMess^ do
    begin
      MessType := tpLongM;
      FCB_ACD  := False;
      FCV_DFC  := False;
      FC       := FCM_CheckLink;
      Data[1]  := $10;
      Data[2]  := $20;
      Data[3]  := $30;
```

```
      Data[4]  := $40;
      Data[5]  := $50;
      LSMess := cSizeHeadSndMsg + 5 {5 je délka pole data};
    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 odpovědi }
      with RMess^ do
      case MessType of
        tpSACK:
          Writeln('Přijat SACK');
        tpSNAK:
          writeln('Přijat SNAK');
        tpShortM:
          begin
            Writeln('Přijata datová zpráva pevné délky')
            for I:=1 to LRMess-cSizeHeadRecMsg do
              Write(Data[I], ' ');
          end;
        tpLongM:
          begin
            Writeln('Přijata datová zpráva proměnné délky')
            for I:=1 to LRMess-cSizeHeadRecMsg do
              Write(Data[I], ' ');
          end;
        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.
```