ChnDF1

JEDNOTKA DEFINUJÍCÍ KOMUNIKAČNÍ PROTOKOL DF1

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)

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

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	16.05.2003	Úprava dokumentu dle ISO9000. Doplnění konstant res_ErrSum, res_ErrMessFrame a MaxTBuf. Doplnění proměnné CH_FlCrc.
1.20	4.XX	Wil	05.04.2004	Doplnění a úprava konstant res_ErrSum, res_ErrFrame a res_ErrLen.

1.2. Účel dokumentu

Tento dokument slouží jako popis jednotky definující komunikační protokol DF1.

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.

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 formát protokolu používaného při komunikaci se zařízeními SLC firmy Allen-Bradley a obstarává zabezpečení dat, vkládání a vyjímání nadbytečností, tak jak to tento protokol předepisuje. Fyzický přenos dat je zajištěn prostřednictvím nižší komunikační vrstvy.

Knihovna ChnDF1 definuje komunikační objekt **tChnDF1**, který je dědicem od rodičovského komunikačního objektu tChnVirt. Instance objektu tChnDF1 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 **tAddChnDF1**, který je dědicem od rodičovského objektu tAddChnVirt. Objekt tAddChnDF1 zajistí, aby daný komunikační objekt (objekt tChnDF1) byl k aplikaci připojen a popřípadě zajistí vytvoření instance tohoto objektu. Po přilinkování této jednotky do aplikace (příkazem "uses ChnDF1"), se jméno objektu tChnDF1 automaticky vloží do seznamu správců komunikačních objektů pro případné použití.

Protože je objekt **tChnDF1** 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 = 'DF1';
```

Konstanta **cName** definuje jméno komunikačního objektu **tChnDF1**.

```
tMessType = (tpAckNak, tpEnq, tpPoll, tpNoPollRep, tpDataMess);
```

Výčtový typ **tMessType** definuje základní typy zpráv.

Typ **tpAckNak** je pro krátké pozitivní či negativní potvrzení od Slave i Master stanice.

Typ **tpEnq** je pro krátkou zprávu pro zopakování odpovědi od Slave i Master stanice.

Typ **tpPoll** je pro krátkou dotazovací zprávu (Polling) na odpověď od Master stanice (pouze při nastavení Half-duplexního režimu protokolu).

Typ **tpNoPollRep** je pro krátkou odpověď od Slave stanice na zprávu Polling, že Slave stanice nemá žádnou odpověď pro Master stanici (pouze při nastavení Half-duplexního režimu protokolu).

Typ **tpDataMess** je pro dlouhou zprávu s datovým polem od Slave i Master stanice.

```
MaxDataLen  = 244;
```

Konstanta **MaxDataLen** definuje maximální délku přenášeného datového bloku v bytech.

```
MaxTBuff    = 65500;
```
Konstanta **MaxTBuff** definuje maximální velikost vysílacího bufferu.

```
Res_ErrFrame      = $20;
```
Konstanta **res_ErrFrame** je definována jako chybný formát zprávy.

```
Res_ErrSum        = $21;
```
Konstanta **res_ErrSum** je definována jako chyba kontrolního součtu BCC.

```
Res_ErrLen        = $22;
```
Konstanta **res_ErrLen** je definována jako chyba délky zprávy.

4.1. Řídící znaky protokolu

```
SOH     = $01;
STX     = $02;
ETX     = $03;
EOT     = $04;
ENQ     = $05;
ACK     = $06;
NAK     = $0F;
DLE     = $10;
```
Konstanta **SOH** definuje kód znaku pro začátek dlouhé zprávy s datovým polem od Master stanice při nastavení Half-duplexního režimu protokolu.
Konstanta **STX** definuje kód znaku pro začátek dlouhé zprávy s datovým polem od Master i Slave stanice při nastavení Full-duplexního režimu protokolu a u Slave stanice i při nastavení Half-duplexního režimu protokolu.
Konstanta **ETX** definuje kód znaku pro konec dlouhé zprávy s datovým.
Konstanta **EOT** definuje kód znaku pro dotazovací zprávu Polling.
Konstanta **ENQ** definuje kód znaku pro požadavek na zopakování odpovědi.
Konstanta **ACK** definuje kód znaku pro krátké pozitivní potvrzení.
Konstanta **NAK** definuje kód znaku pro krátké negativní potvrzení.
Konstanta **DLE** definuje kód transparentního znaku pro odlišení datových a řídících znaků protokolu. Řídící znaky protokolu se vysílají s předřazeným znakem DLE (např. DLE ACK), přičemž znak DLE se nezapočítává do kontrolního součtu zprávy. Datový znak mající stejnou hodnotu jako znak DLE se vysílá zdvojeně (DLE DLE) a při počítání kontrolního součtu pro něj platí stejná pravidla. Pokud ale některý z vysílaných znaků kontrolního součtu (CRC/BCC) je roven hodnotě některého řídícího znaku nebo znaku DLE, neposílá se ani jako zdvojený ani s předřazeným DLE.

4.2. Kódy služeb protokolu a příslušné masky

Následující konstanty určují kód služby (CF - Command & Function) dlouhé zprávy s datovým polem.

Význam jednotlivých bitů CF:

bit	15	14	13	12	11..8	7..0
popis	0	Reply	Priority	0	Command	Function

0 - normální priorita příkazu { maska $2000 }
1 - vysoká priorita příkazu

0 - příkazová zpráva { maska $4000 }
1 - odpověď na zprávu

Tomuto rozložení odpovídají i tyto masky:

```
MskCF_Reply          = $4000;
MskCF_HiPriority     = $2000;
MaskCmd              = $0F00;
```

Konstanta **MskCF_Reply** definuje masku pro nastavení či otestování 14.bitu v CF (6.bit v Command), že se jedná o odpověď (nastavují převážně Slave stanice).

Konstanta **MskCF_HiPriority** definuje masku pro nastavení či otestování 13.bitu v CF (5.bit v Command), že se jedná o zprávu s vysokou prioritou (většinou se ale nepoužívá).

Konstanta **MaskCmd** definuje masku pro vymaskování přebytečných bitů z CF k získání holého Command.

```
CF_Echo              = $0600;
CF_ReadDiagnCounts   = $0601;
CF_SetVariables      = $0602;
CF_DiagnosticSts     = $0603;
CF_SetTimeOut        = $0604;
CF_SetNAKs           = $0605;
CF_SetENQs           = $0606;
CF_ResetDiagnCounts  = $0607;
CF_SetDataTableSize  = $0608;
CF_ReadLinkParams    = $0609;
CF_SetLinkParams     = $060A;
CF_DisableOutputs    = $0700;
CF_EnableOutputs     = $0701;
CF_EnablePLCScaning  = $0703;
CF_EnterDnLoadMode   = $0704;
CF_ExitDnUpLoadMode  = $0705;
CF_EnterUpLoadMode   = $0706;
CF_ShutDown          = $0F07;
CF_SetCPUMode        = $0F3A;
CF_ChangeMode1       = $0F3A;
CF_ChangeMode2       = $0F80;
CF_OpenFile          = $0F81;
CF_CloseFile         = $0F82;
CF_WordRangeWrite    = $0F00;
CF_WordRangeRead     = $0F01;
CF_WriteFile         = $0F03;
CF_ReadFile          = $0F04;
CF_WriteBytesPhysical= $0F18;
CF_ApplyPortConfg    = $0F8F;
CF_DisableForces     = $0F41;
CF_TypedRead         = $0F68;
CF_TypedWrite        = $0F67;
CF_DnLoadRequest     = $0F05;
CF_UpLoad            = $0F06;
CF_DnLoadAllRequest  = $0F50;
CF_DnLoadCompleted   = $0F52;
CF_UpLoadAllRequest  = $0F53;
CF_UpLoadCompleted   = $0F55;
CF_GetEditSource     = $0F11;
CF_InitMemory        = $0F57;
CF_ModifyPLC2File    = $0F5E;
CF_ProtectWrite      = $0000;
CF_UnProtectRead     = $0100;
CF_UnProtectWrite    = $0800;
CF_WriteBit          = $0F02;
CF_ProtectWriteBit   = $0200;
CF_UnProtectWriteBit = $0500;
CF_PhysicalWrite     = $0F08;
CF_PhysicalRead2     = $0F09;
```

```
CF_PhysicalRead1     = $0400;
CF_ProtectReadTFile  = $0FA7;
CF_ProtectWriteTFile = $0FAF;
CF_ProtectRead3Addr  = $0FA2;
CF_ProtectWrite3Addr = $0FAA;
CF_ReadBytesPhysical = $0F17;
CF_ReadModifyWrite   = $0F26;
CF_ReadModifyWriteN  = $0F79;
CF_ReadSectionSize   = $0F29;
CF_RestartRequest    = $0F0A;
CF_ReturnEditSource  = $0F12;
```

Pozn.: U kódů CF_PhysicalRead1, CF_ProtectWriteBit, CF_ProtectWrite, CF_UnProtectWriteBit, CF_UnProtectRead a CF_UnProtectWrite se nepoužívá jejich spodní část (Function).

Pozn.: Význam jednotlivých CF je poplatný každé datové zprávě a různá zařízení podporují jen některé z nich. Více viz dokumentace o protokolu DF1 vydávaná firmou Allen-Bradley.

4.3. Struktury přijímacích a vysílacích bufferů

```
pSendRecord = ^tSendRecord;
tSendRecord = record
  case MessType   : tMessType of
    tpAckNak  :
      (FlAck      :boolean;);
    tpEnq,
    tpPoll,
    tpNoPollRep:
      ();
    tpDataMess:
      (Dummy1_1   : byte;
       case CF    : word of
         CF_WriteBytesPhysical:
           (Sts   : byte;
            Dummy1_2: byte;
            Tns   : word;
            Data  : array[0..MaxDataLen-1]of byte;);
      );
end;
```

TSendRecord je typ variantního záznamu, který svou strukturou odpovídá datům protokolu DF1 posílaným Master i Slave stanicí, přičemž je zbaven nadbytečností, které jsou při vysílání doplněny. Položka **MessType** udává typ zprávy. Při krátké potvrzovací zprávě se dále používá položka **FlAck**, která určuje příznak pozitivního potvrzení. Při posílání dlouhých zpráv s datovým polem se dále používá položka **CF** určující kód (Command & Function) služby, položka **Sts** určující status, položka **Tns** určující číslo funkční zprávy a položka **Data**, která definuje pole dat. Položky **Dummy1_1** a **Dummy1_2** nemá žádný zvláštní význam, v záznamu jsou definovány pouze pro zarovnání jiných položek na adresu dělitelnou čtyřma či dvěma.

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

TRecRecord je typ variantního záznamu, který svou strukturou odpovídá datům protokolu DF1 přijímaným Master či Slave stanicí, přičemž je zbaven nadbytečností, které jsou při příjmu odstraněny. Svou vnitřní strukturou je shodný s type **TSendRecord**.

5. Objekty

5.1. tChnDF1

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_FlCrc : Boolean;`
Položka **CH_FlCrc** definuje příznak používání kontrolního součtu CRC16 místo BCC.

`CH_RSumBcc : tBcc8;`
Položka **CH_RSumBcc** se používá pro počítání 8-mi bitového kontrolního součtu přijímače.

`CH_SSumBcc : tBcc8;`
Položka **CH_SSumBcc** se používá pro počítání 8-mi bitového kontrolního součtu vysílače.

`CH_RSumCrc : tCrc16;`
Položka **CH_RSumCrc** se používá pro počítání 16-ti bitového kontrolního součtu přijímače.

`CH_SSumCrc : tCrc16;`
Položka **CH_SSumCrc** se používá pro počítání 16-ti bitového 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).

`CH_FullDup : Boolean;`
Položka **CH_FullDup** definuje příznak používání Full-Duplexního režimu protokolu. V opačném případě se používá Half- Duplexní režim protokolu.

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_Master  := true;
  CH_FullDup := true;
  CH_FlCrc   := true;
  CH_DLE     := false;
  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 tChnTecom 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).

FHD=Full / Half

Parametrem **FHD** ("Full / Half Duplex Protokol") se volí režim protokolu.

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.

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**.

CRC=On / Off

Parametrem **CRC** ("Using Crc16 or Bcc8") se určí příznak používání 16-ti bitového součtu bloku dat cyklickým polynomem.

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 DF1.

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 DF1 na podkladu záznamu typu tSendRecord, na který ukazuje parametr Buff. Parametr Len udává délku vysílacího bufferu pro vysílání. Tento parametr je nutno správně zadat v případě dlouhé zprávy s daty a to následovně: Len = počet vysílaných bytů v poli Data + 8. V případě jiné zprávy (bez datového pole či krátkého potvrzení) 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 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 DF1.

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. tAddChnDF1

Typ **tAddChnDF1** je typem objektu, který slouží k definování prvku v seznamu správců komunikačních objektů (tzv. správce komunikačního objektu tChnDF1 v seznamu správců). Objekt tAddChnDF1 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 tChnDF1 a ukazatel na instanci tohoto objektu vrací jako svoji funkční hodnotu.

6. Struktura zpráv protokolu DF1

Protokol DF1 má dva režimy: Full-Duplexní a Half-Duplexní. Nastavení režimu protokolu se provádí voláním metody ChSetParam. Struktury a principy posílání zpráv pro oba režimy se liší.

6.1. Half-Duplexní režim protokolu

Je využíván pro komunikace mezi jednou nadřazenou stanicí (Master) a jednou či více podřazenými stanicemi (Slave) s jedinečnou adresou (Node). Jednotlivé Slave stanice mohou mezi sebou komunikovat prostřednictvím Master stanice následovně: Master stanice se dotáže Slave stanice zprávou Polling a pokud Slave stanice odpoví na tuto zprávu s jinou cílovou adresou (DNode), než má Master, přetransformuje Master stanice tuto zprávu do svého formátu a pošle jí příslušné Slave stanici. Tato komunikace mezi Slave stanicemi závisí na konkrétní aplikaci a na nastavení Master stanice.

Zpráva s datovým polem vysílaná stanicí **Master**.

SOH	DNO	STX	DNODE	NODE	CMD	STS	TNS	DATA	ETX	BCC/CRC

Zpráva s datovým polem vysílaná stanicí **Slave**.

STX	DNODE	NODE	CMD	STS	TNS	DATA	ETX	BCC/CRC

Krátké pozitivní potvrzení od stanice **Master** i **Slave**.

ACK

Zpráva Polling (dotaz na data) od stanice **Master**.

ENQ	DNO	BCC

Odpověď na zprávu Polling od stanice **Slave**, že Slave stanice nemá žádná data pro stanici Master či jinou Slave stanici.

EOT

Význam položek:

Řídící znaky SOH, STX, ETX, ACK, NAK, ENQ a EOT se vysílají samozřejmě s předřazeným znakem DLE.

U zprávy Polling se jako kontrolního součtu používá vždy BCC.

DNO - adresa cílové stanice (1byte)
DNODE - adresa cílové stanice (1byte)
NODE - adresa zdrojové stanice (1byte)
CMD - Command - horní byte z CF (1byte)
STS - status zprávy (1byte)
TNS - řídící číslo zprávy (1word)
DATA - pole dat, kde první byte je obvykle FNC (Function) - spodní byte z CF, mimo těch CF, které Function nepoužívají
BCC - bitový doplněk kontrolního součtu zvětšený o 01h (1byte) počítaný v případě zprávy s datovým polem z položek DNO, DNODE, NODE, CMD, STS, TNS a všech bytů pole DATA, v případě zprávy Polling je roven položce DNO.

CRC - 16-ti bitový zbytek po dělení cyklickým polynomem $X^{16} + X^{15} + X^2 + 1$ (1word) počítaný ze stejných položek jako BCC

6.1.1. Linková vrstva protokolu v Half-Duplexním režimu

1. Master stanice posílá zprávu s datovým polem Slave stanici.
2. Pokud Slave stanice zprávu správně přijala a dekódovala, odpovídá Master stanici krátkým pozitivním potvrzením ACK nebo v případě chybného přijetí zprávy neodpovídá.
3. Pokud Master stanice nepřijala do stanovené doby pozitivní potvrzení ACK, měla by datovou zprávu zopakovat se stejným řídícím číslem zprávy TNS. V opačném případě by měla poslat Slave stanici zprávu Polling.
4. Pokud Slave stanice zprávu Polling správně přijala, odpovídá Master stanici datovou zprávou, nebo v případě, že nemá pro Master stanici žádná data, odpovídá zprávou EOT.
5. Pokud Master stanice správně přijala a dekódovala datovou zprávu od Slave stanice vyšle Slave stanici krátké pozitivní potvrzení ACK.

Př. Master stanice pošle Slave stanici tři různé datové zprávy (každou s jiným řídícím číslem zprávy TNS) a po vyslání každé z nich počká na krátkou pozitivní odpověď ACK. Slave stanice všechny tři zprávy správně přijme a dekóduje a odpoví na každou z nich krátkým pozitivním ACK. Skutečné datové odpovědi má uložené ve svém bufferu odkud je Master stanice získá vysláním tří zpráv Polling s čekáním na danou datovou odpověď a (po jejím přijetí) s následným pozitivním potvrzením ACK. Každá datová odpověď od Slave stanice bude mít řídící číslo zprávy TNS shodné s řídícím číslem zprávy datové zprávy od Master stanice vysílané na začátku tohoto příkladu. Pokud by Master stanice vyslala nakonec čtvrtou zprávu Polling, dostala by odpověď EOT, protože Slave stanice by už neměla ve svém bufferu další datové odpovědi. Pokud by Master stanice neodpověděla na datovou odpověď Slave stanici krátkým pozitivním potvrzením ACK, Slave stanice by tuto datovou odpověď ze svého bufferu neodmazala a odpověděla by na další zprávu Polling Master stanici touto samou datovou odpovědí.

6.2. Full-Duplexní režim protokolu

Je využíván pro komunikace bod to bod.

Zpráva s datovým polem vysílaná stanicí **Master**.

STX	DNODE	NODE	CMD	STS	TNS	DATA	ETX	BCC/CRC

Zpráva s datovým polem vysílaná stanicí **Slave**.

STX	DNODE	NODE	CMD	STS	TNS	DATA	ETX	BCC/CRC

Krátké pozitivní potvrzení od stanice **Master** i **Slave**.

ACK

Krátké negativní potvrzení od stanice **Master** i **Slave**.

NAK

Krátká zpráva pro zopakování zprávy či odpovědi od stanice **Master** i **Slave**.

ENQ

Význam položek:

Řídící znaky STX, ETX, ACK, NAK, ENQ a EOT se vysílají samozřejmě s předřazeným znakem DLE.

DNODE - adresa cílové stanice (1byte)
NODE - adresa zdrojové stanice (1byte)
CMD - Command - horní byte z CF (1byte)
STS - status zprávy (1byte)
TNS - řídící číslo zprávy (1word)
DATA - pole dat, kde první byte je obvykle FNC (Function) - spodní byte z CF, mimo těch CF, které Function nepoužívají
BCC - bitový doplněk kontrolního součtu zvětšený o 01h (1byte) počítaný v případě zprávy s datovým polem z položek DNODE, NODE, CMD, STS, TNS a všech bytů pole DATA.
CRC - 16-ti bitový zbytek po dělení cyklickým polynomem $X^{16} + X^{15} + X^2 + + 1$ (1word) počítaný ze stejných položek jako BCC

6.2.1. Linková vrstva protokolu v Full-Duplexním režimu

1. Master stanice posílá zprávu s datovým polem Slave stanici.
2. Pokud Slave stanice zprávu správně přijala a dekódovala, odpovídá Master stanici krátkým pozitivním potvrzením ACK nebo v případě chybného přijetí zprávy odpovídá krátkým negativním potvrzením NAK nebo neodpovídá.
3. Pokud Master stanice nepřijala do stanovené doby pozitivní potvrzení ACK, měla by datovou zprávu zopakovat se stejným řídícím číslem zprávy TNS. V opačném případě čeká na datovou odpověď od Slave stanice.
4. Pokud Slave stanice pošle Master stanici datovou odpověď, kterou Master stanice v pořádku přijme a dekóduje, odpoví Master stanice krátkým pozitivním potvrzením ACK. Pokud Master stanice datovou odpověď dekóduje špatně, odpovídá Slave stanici krátkým negativním potvrzením NAK a Slave stanice pošle datovou odpověď znova. Pokud Master stanice neodpoví na datovou odpověď ani zprávou ACK či NAK pošle Slave stanice Master stanici výzvu ENQ, na kterou by Master stanice měla odpovědět zprávou ACK či NAK.

6.3. Struktura konkrétních zpráv

V následujících odstavcích jsou popsány způsoby nastavování jednotlivých položek ve strukturách vysílacích bufferů pro konkrétní příklady zpráv protokolu DF1. Jako odpovědi od Slave stanice jsou uvedeny už přímo datové odpovědi, linkové zprávy ACK, NAK, Polling apod. zde nejsou uvedeny (jejich princip byl uveden výše).

6.3.1. Zpráva Echo - test připojení PLC

Master stanice vysílá datovou zprávu s nastavenými položkami:

```
MessType := tpDataMess;
CF       := CF_Echo;
Sts      := 0;
Tns      := ... např. 10
Data[0..3]:= 'Ahoj';
```

Délka vysílané zprávy se nastaví na 12 ($8_{hlavička}+4_{data}$).

Slave stanice odpovídá datovou zprávou s nastavenými položkami:

```
MessType := tpDataMess;
CF       := CF_Echo or MskCF_Reply;
Sts      := 0;
Tns      := ... např. 10
Data[0..3]:= 'Ahoj';
```

Délka vysílané zprávy se nastaví na 12 ($8_{hlavička}+4_{data}$).

6.3.2. Zpráva DiagnosticStatus - čtení diagnostických informací

Master stanice vysílá datovou zprávu s nastavenými položkami:

```
MessType := tpDataMess;
CF       := CF_DiagnosticSts;
Sts      := 0;
Tns      := ... např. 11;
```

Délka vysílané zprávy se nastaví na 8 ($8_{hlavička}+0_{data}$).

Slave stanice odpovídá datovou zprávou s nastavenými položkami:

```
MessType := tpDataMess;
CF       := CF_DiagnosticSts or MskCF_Reply;
Sts      := 0;
Tns      := ... např. 11;
Data     := ...
```

Délka vysílané zprávy se nastaví na 8+velikost Data (max.244).

6.3.3. Zpráva ReadDiagnosticCounters - čtení diagnostických čítačů

Master stanice vysílá datovou zprávu s nastavenými položkami:

```
MessType := tpDataMess;
CF       := CF_ReadDiagnCounts;
Sts      := 0;
Tns      := ... např. 12;
Data[0,1]:= Addr např. 0000h
Data[2  ]:= Size např. 10 (max. 244)
```

Délka vysílané zprávy se nastaví na 11 ($8_{hlavička}+3_{data}$).

Slave stanice odpovídá datovou zprávou s nastavenými položkami:

```
MessType := tpDataMess;
CF       := CF_ReadDiagnCounts or MskCF_Reply;
Sts      := 0;
Tns      := ... např. 12;
Data     := ...
```

Délka vysílané zprávy se nastaví na 8+velikost Data (max.244).

6.3.4. Zpráva ResetDiagnosticCounters - vynulování diagnostických čítačů

Master stanice vysílá datovou zprávu s nastavenými položkami:

```
MessType := tpDataMess;
CF       := CF_ResetDiagnCounts;
Sts      := 0;
Tns      := ... např. 13;
```

Délka vysílané zprávy se nastaví na 8 ($8_{hlavička}+0_{data}$).

Slave stanice odpovídá datovou zprávou s nastavenými položkami:

```
MessType := tpDataMess;
CF       := CF_ResetDiagnCounts or MskCF_Reply;
Sts      := 0;
Tns      := ... např. 13;
```

Délka vysílané zprávy se nastaví na 8 ($8_{hlavička}+0_{data}$).

6.3.5. Zpráva Protected Typed Logical Read With Three Address Fields - chráněné čtení z logické adresy složky

Pouze pro SLC 500, SLC 5/03, SLC 5/04.

Master stanice vysílá dlouhou zprávu s nastavenými položkami:

```
MessType := tpDataMess;
CF       := CF_ProtectRead3Addr;
Sts      := 0;
Tns      := ... např. 14;
Data[0]  := Size        např. 5;
Data[1]  := File No.    např. 07h;
Data[2]  := File Type   např. 89h; {integer file}
Data[3]  := Element No. např. 0;
Data[4]  := Sub-El. No. např. 0;
```

Délka vysílané zprávy se nastaví na 13 ($8_{hlavička}+5_{data}$).

Slave stanice odpovídá datovou zprávou s nastavenými položkami:

```
MessType := tpDataMess;
CF       := CF_ProtectRead3Addr or MskCF_Reply;
Sts      := 0;
Tns      := ... např. 14;
Data     := ...
```

Pokud by Sts byl různý od 0, representoval by poslední byte pole Data rozšířený kód statusu (Extended Status).

Délka vysílané zprávy se nastaví na 8+ velikost Data.

6.3.6. Zpráva Protected Typed Logical Write With Three Address Fields - chráněný zápis na logickou adresu složky

Master stanice vysílá dlouhou zprávu s nastavenými položkami:

```
MessType := tpDataMess;
CF       := CF_ProtectWrite3Addr;
Sts      := 0;
Tns      := ... např. 15;
Data[0]  := Size        např. 5;
Data[1]  := File No.    např. 07h;
Data[2]  := File Type   např. 89h; {integer file}
Data[3]  := Element No. např. 0;
Data[4]  := Sub-El. No. např. 0;
```

```
Data[5..9]:= ...
```

Délka vysílané zprávy se nastaví na 13 ($8_{hlavička}+5_{hlavi4ka\ data}+5_{skutečná\ data}$).
Slave stanice odpovídá opět dlouhou zprávou s nastavenými položkami:

```
MessType := tpDataMess;
CF       := CF_ProtectWrite3Addr or MskCF_Reply;
Sts      := 0;
Tns      := ... např. 15;
Data[0]  := ...
```

Pokud by Sts byl různý od 0, representoval by byte pole Data[0] rozšířený kód statusu (Extended Status).

Délka vysílané zprávy se nastaví na 8 ($8_{hlavička}+0_{data}$).

7. Příklad

Následující příklad ukazuje způsob vyslání zprávy Echo do SLC Allen-Bradley pro otestování připojení tohoto zařízení. Fyzický přenos se realizuje prostřednictvím knihovny ChnCom.

```
uses
  uString,
  ChnVirt,
  ChnCom,
  ChnDF1;

const
  ParamStr : tParamStr =
             'NAM=DF1 LSB=500 NOD=1 DNO=2 FHD=FULL CRC=ON' +
             'NAM=COM COM=1 IRQ=4 BD=9600 BIT=8 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(ChnDF1.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:=tpDataMess;
  CF      :=CF_Echo;
  Sts     :=0;
  Tns     :=0;
  Data[0] :=Ord('A');
  Data[0] :=Ord('h');
  Data[0] :=Ord('o');
  Data[0] :=Ord('j');
  LSMess:=8+4;
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 ACK }
  with RMess^ do
  begin
    if (MessType = tpAckNak) and
       (FlAck    = true    ) then
    begin
      { 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 datové odpovědi na zprávu Echo }
        if (MessType = tpDataMess) and
           (CF = CF_Echo or MskCF_Reply) and
           (Tns = 0) and
           (Sts = 0) and
           (LRMess = 8+5) then
          writeln('Ok')
        else
          writeln('Chyba');
    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.
```