ChnCom

JEDNOTKA SÉRIOVÉ KOMUNIKACE RS232 / RS485 S OBVODEM I8250

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 1.40

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: 23.09.2005
Datum posledního uložení dokumentu: 23.09.2005
(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	1.XX	Tu	22.05.2003	Úprava dokumentu dle ISO9000.
1.20	4.5X	Wil	14.04.2004	Upravení popisu. Přidány konstanty res_ErrRecBuffOvf, res_ErrOverrun, res_ErrParity, res_ErrFraming, res_BreakInt.
1.30	4.6X	Wil	05.04.2005	Přidána obsluha FIFO. Přidány čítače počtu přerušení. Přidána detekce nefungujícího IRQ při vysílání.
1.40	4.7X	Wil	23.09.2005	Upraven popis pro nastavování IRQ.

1.2. Účel dokumentu

Tento dokument slouží jako popis jednotky sériové komunikace RS232 / RS485 s obvodem i8250.

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 rodičovský prvek komunikačních objektů. Dále s manuálem ChnTypes definujícím typy a konstanty pro komunikační knihovny.

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 ChnCom definuje objekt **tChnCom**, jehož instance vytváří fyzickou vrstvu v komunikačním kanálu tvořenou sériovým rozhraním RS232 nebo RS485 s obvodem i8250. Ke své činnosti využívá přerušovacího systému počítače. Znaky přicházející po komunikační lince jsou v přerušovací proceduře ukládány do vstupního kruhového vyrovnávacího bufferu, z kterého jsou předávány metodami **ChReceive** a **ChReceiveChar** k dalšímu zpracování. Znaky určené k odeslání jsou zasílány z výstupního bufferu rovněž s využitím přerušovacího systému.

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

Protože je objekt **tChnCom** 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 jednoduché 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 = 'COM';
```

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

4.1. Konstanty statusu přijatého znaku

Následující chyby vrací metoda **ChReceiveResult** po příjmu znaku metodou **ChReceiveChar**. Jelikož v některých případech může nastat i více chyb najednou, definují tyto konstanty tzv. bitové masky pro spodní byte funkční hodnoty metody ChReceiveResult.

```
res_ErrRecBuffOvf = $0080;
```

Přijatý znak způsobil přetečení přijímacího bufferu – buffer byl vymazán.

```
res_ErrOverrun    = ChnTypes.MErrOvr;
```

Maska pro dekódování chyby Overrun.

```
res_ErrParity     = ChnTypes.MErrPar;
```

Maska pro dekódování chyby Parity

```
res_ErrFraming    = ChnTypes.MErrFrm;
```

Maska pro dekódování chyby Framing.

```
res_BreakInt      = ChnTypes.MBrkInt;
```

Maska pro dekódování Break Interrupt.

Následující chyby vrací metoda **ChSendResult** po metodách **ChSend, ChSendReady, ChSendTick**.

```
res_ErrSendTimeout      = $0030;
```
Timeout při vysílání (nechodí přerušení).

Následující chyby vrací metoda **ChResult** po metodě **ChOpen**.

```
res_ErrFIFOunavailable = $0040;
```
Vyžaduje se FIFO, které ale není k dispozici.

4.2. Konstanty kódů příkazů pro metodu ChGetBinParam

```
cmd_GetMSR   = $0101;
```
Konstanta definuje kód příkazu pro vrácení stavu všech modemových signálů (Modem Status Registeru).

```
cmd_GetDCTS  = $0102;
```
Konstanta definuje kód příkazu pro vrácení stavu modemového signálu DCTS v 0.bitu výsledku funkce.

```
cmd_GetDDSR  = $0103;
```
Konstanta definuje kód příkazu pro vrácení stavu modemového signálu DDSR v 0.bitu výsledku funkce.

```
cmd_GetTERI  = $0104;
```
Konstanta definuje kód příkazu pro vrácení stavu modemového signálu TERI v 0.bitu výsledku funkce.

```
cmd_GetDRLSD = $0105;
```
Konstanta definuje kód příkazu pro vrácení stavu modemového signálu DRLSD v 0.bitu výsledku funkce.

```
cmd_GetCTS   = $0106;
```
Konstanta definuje kód příkazu pro vrácení stavu modemového signálu CTS v 0.bitu výsledku funkce.

```
cmd_GetDSR   = $0107;
```
Konstanta definuje kód příkazu pro vrácení stavu modemového signálu DSR v 0.bitu výsledku funkce.

```
cmd_GetRI    = $0108;
```
Konstanta definuje kód příkazu pro vrácení stavu modemového signálu RI v 0.bitu výsledku funkce.

```
cmd_GetRLSD  = $0109;
```
Konstanta definuje kód příkazu pro vrácení stavu modemového signálu RLSD v 0.bitu výsledku funkce.

```
cmd_GetCntIrq   = $0110;
```
Konstanta definuje kód příkazu pro vrácení počtu všech vyvolaných přerušení.

```
cmd_GetCntRecB  = $0111;
```
Konstanta definuje kód příkazu pro vrácení počtu přerušení způsobených příjmem znaku nebo statusem.

```
cmd_GetCntSendB = $0112;
```
Konstanta definuje kód příkazu pro vrácení počtu přerušení způsobených odvysíláním znaku.

```
cmd_GetFIFOenabled = $0120;
```
Konstanta definuje kód příkazu pro vrácení příznaku, je-li skutečně povoleno FIFO (volat po metodě ChOpen).

4.3. Konstanty kódů příkazů pro metodu ChSetBinParam

```
cmd_SetMCR   = $0101;
```
Konstanta definuje kód příkazu pro nastavení všech modemových signálů (Modem Control Registeru) dle nejnižšího byte parametru funkce Param.

```
cmd_SetDTR   = $0102;
```
Konstanta definuje kód příkazu pro nastavení modemového signálu DTR dle 0.bitu parametru funkce Param.

```
cmd_SetRTS   = $0103;
```
Konstanta definuje kód příkazu pro nastavení modemového signálu RTS dle 0.bitu parametru funkce Param.

4.4. Konstanty a typy přijímacích a vysílacích bufferů

```
tTChar    = byte;
```
Typ pro položku vysílacího bufferu.

```
tRChar    = record
  Hod : byte;
  Sts : byte;
end;
```
Typ pro položku přijímacího bufferu.

```
MaxRBuf   = 65500 div SizeOf(tRChar);
```
Maximální velikost přijímacího bufferu (32750).

```
MaxTBuf   = 65500 div SizeOf(tTChar);
```
Maximální velikost vysílacího bufferu (65500).

```
tAByte    = array[0..65500] of Byte;
```
Typ pole byte.

```
pAByte    = ^tAByte;
```
Typ ukazatele na pole byte.

```
tTBuff    = array[0..MaxTBuf] of tTChar;
```
Typ vysílacího bufferu.

```
tRBuff    = array[0..MaxRBuf] of tRChar;
```
Typ přijímacího bufferu.

```
pTBuff    = ^tTBuff;
```
Typ ukazatele na vysílací buffer.

```
pRBuff    = ^tRBuff;
```
Typ ukazatele na přijímací buffer.

4.5. Konstanty a typy pro počet obsluhovaných COM portů

```
MaxCom    = 4;
```
Maximální počet obsluhovaných sériových portů. Hodnota této konstanty je závislá na počtu definovaných interních datových struktur v této knihovně pro obsluhu přerušení od obvodu i8250.

```
tCtCom    = 1..MaxCom;
```
Intervalový typ počtu sériových portů.

5. Proměnné

```
AChnCom : array [tCtCom] of pChnCom = (nil,nil,nil,nil);
```

Pole ukazatelů na komunikační objekty tChnCom pro obsluhu obvodu i8250.

6. Objekty

6.1. tChnCom

6.1.1. Položky

```
CH_Cislo    : Byte;
```

Položka **CH_Cislo** určuje číslo sériového kanálu. Jen pro vnitřní použití.

```
CH_Addr     : Word;
```

Položka **CH_Addr** obsahuje adresu sériového komunikačního adaptéru i8250.

```
CH_Irq      : tIrq;
```

Položka **CH_Irq** obsahuje číslo přerušení IRQ, na kterém adaptér i8250 žádá o přerušení.

```
CH_Rate     : tRate;
```

Položka **CH_Rate** obsahuje požadovanou přenosovou rychlost v bitech za sekundu.

```
CH_Parity   : tParity;
```

Položka **CH_Parity** obsahuje požadovanou paritu přenášeného znaku.

```
CH_Stop     : tStop;
```

Položka **CH_Stop** obsahuje počet stop bitů přenášeného znaku.

```
CH_Length   : tLength;
```

Položka **CH_Length** obsahuje počet datových bitů přenášeného znaku.

```
CH_RSDelay1 : Longint;
```

Položka **CH_RSDelay1** definuje minimální časovou prodlevu v ms před vysíláním.

```
CH_RSDelay2 : Longint;
```

Položka **CH_RSDelay2** definuje minimální časovou prodlevu v ms po vysílání.

```
CH_RecOn    : Boolean;
```

Položka **CH_RecOn** určuje, zda se má během vysílání nechat povolený příjem znaků či zda ho zakázat.

```
CH_RecDel   : Boolean;
```

Položka **CH_RecDel** určuje, zda se má (při nastaveném zakázání příjmu během vysílání) vypustit první přijatý znak po skončeném vysílání.

```
CH_FifoEnb  : Boolean;
```

Položka **CH_FifoEnb** určuje, zda se má používat přijímací a vysílací FIFO.

```
CH_FifoErr  : Boolean;
```

Položka **CH_FifoErr** určuje, zda se má hlásit chyba, když daný komunikační obvod i8250 neumožňuje FIFO a aplikace ho vyžaduje.

```
CH_STick    : Boolean;
```

Položka **CH_STick** je určena jen pro vnitřní použití, pro určení, zda je vykonávána činnost vysílacího automatu.

```
CH_FlDTR    : Boolean;
```

Položka **CH_FlDTR** definuje výstupní signál DTR (hodnota True odpovídá nastaví signálu na hodnotu 1).

```
CH_FlRTS    : Boolean;
```

Položka **CH_FlRTS** definuje výstupní signál RTS (hodnota True odpovídá nastaví signálu na hodnotu 1).

6.1.2. Metody

6.1.2.1. Init konstruktor

```
constructor Init;
```

Konstruktor **Init** slouží k vytvoření a inicializaci instance komunikačního objektu. Ve svém těle nejdříve zavolá zděděný konstruktor **Init** (inherited Init) z rodičovského objektu tChnVirt a poté inicializuje položky objektu. Tělo konstruktoru vypadá následovně:

```
  inherited Init;
  CH_Type     := cName;
  CH_Name     := CH_Type;
  CH_NumName  := ChNumName(CH_Type);
  CH_Cislo    := 1;
  CH_Addr     := ACom1;
  CH_Irq      := Irq4;
  CH_Rate     := 9600;
  CH_Parity   := ParOdd;
  CH_Stop     := Stop1;
  CH_Length   := Bits8;
  CH_RSDelay1 := 0;
  CH_RSDelay2 := 0;
  CH_RecOn    := true;
  CH_RecDel   := false;
  CH_FifoEnb  := false;
  CH_FifoErr  := false;
  CH_STick    := false;
  CH_FlDTR    := true;
  CH_FlRTS    := false;
```

6.1.2.2. 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**.

6.1.2.3. Done destruktor

```
destructor  Done;
```

Destruktor **Done** slouží ke zrušení instance komunikačního objektu. Pokud je v paměti alokován přijímací buffer, bude odstraněn a poté je zavolán zděděný destruktor **Done** z objektu tChnVirt (inherited Done).

6.1.2.4. 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** (zděděná metoda ChSetParam od svého předka). Metoda **ChSetOneParam** komunikačního objektu tChnCom dekóduje tyto parametry:

COM=aaa

Parametr **COM** určuje číslo sériového kanálu COM. aaa může nabývat hodnot 1 až 8. Adresy odpovídající jednotlivým kanálům COM jsou uvedeny v jednotce ChnTypes.

ADD=bbb

Parametr **ADD** "Addres" určuje adresu sériového kanálu COM. Použije se jen v případě, kdy komunikační adaptér i8250 není na standardní vstupně/výstupní adrese (viz parametr COM). bbb může nabývat hodnot 0 až $ffff.

IRQ=ccc

Parametr **IRQ** určuje číslo přerušení IRQ, na kterém adaptér i8250 žádá o zpracování přerušení. Hodnota ccc je závislá na použitém typu procesoru. Výčet použitelných IRQ definuje intervalový typ tIRQ v knihovně Hwsyst.

BD =ddd

Parametr **BD** "BaudRate" určuje přenosovou rychlost požadované sériové komunikace. ddd může nabývat hodnot 25 až 115200 Bd.

BIT=eee

Parametr **BIT** "Number of Data Bits" určuje počet datových bitů v přenášeném znaku. eee může nabývat hodnot 5 až 8.

PAR=fff

Parametr **PAR** "Parity" určuje paritu přenášeného znaku. fff může nabývat hodnot 0 "Odd" pro lichou paritu, E "Even" pro sudou paritu a N "None" pro znak bez parity.

STO=ggg

Parametr **STO** "Number of Stop Bits" určuje počet stop-bitů v přenášeném znaku. ggg může nabývat hodnot 1 nebo 2.

LRB=hhh

Parametr **LRB** "Length of Receive Buffer" určuje velikost přijímacího kruhového vyrovnávacího bufferu. Buffer je alokován na heapu a každá jeho položka zaujímá v paměti prostor o velikosti 2 byte (přijatý znak a status). Platí, čím větší je vstupní buffer, tím více znaků dokáže udržet, aniž by byly znaky metodami **ChReceive** a **ChReceiveChar** odebírány. Velikost bufferu je shora omezena na 32750. Je proto nutno volit kompromis mezi velikostí bufferu a periodou, kterou přijaté znaky odebíráme. Tento parametr je povinný. Pokud by zadán nebyl, vrátil by ChResult po ChConnect chybu res_ErrNoRecBuff. Pokud se přijímací buffer nadefinuje malý, tj. COM přijme v přerušení větší množství znaků, než aplikace zpracuje metodami ChReceive nebo ChReceiveChar, dojde k jeho přetečení. V tomto případě se

obsah celého bufferu vymaže a poslední přijatý znak, který způsobil jeho přetečení, je do bufferu uložen se statusem res_ErrRecBuffOvf.

RS1=iii

Parametr **RS1** určuje zpoždění v ms před vysíláním.

RS2=jjj

Parametr **RS2** určuje zpoždění v ms po vysílání.

REC=ON / OFF / OFF2

Parametr **REC** "Receive While Sending" určuje, zda má být pří vysílání povolen příjem znaků. Hodnota **ON** znamená, že příjem znaků je povolen i během vysílání. Hodnota **OFF** znamená, že během vysílání je příjem znaků potlačen (pro RS232 a RS485 - 4drát). Pro RS422 - 2drát se poslední vysílaný znak zprávy automaticky zpětně přijímá. Proto hodnota **OFF2** má stejný význam jako hodnota OFF, ale navíc se vypouští první zpětně přijatý znak po odvysílání zprávy.

DTR=ON / OFF

Parametr **DTR** "Setting of DTR Signal" určuje, zda má být nastaven výstupní signál DTR na hodnotu 1 (ON) či na hodnotu 0 (OFF).

RTS=ON / OFF

Parametr **RTS** "Setting of RTS Signal" určuje, zda má být nastaven výstupní signál RTS na hodnotu 1 (ON) či na hodnotu 0 (OFF).

FIF=ON / OFF / IFAVAIL

Parametr **FIF** "FIFO" určuje, zda se má používat vysílací a přijímací FIFO. Hodnota **ON** znamená zapnutí FIFO, přičemž pokud daný obvod i8250 FIFO neumožňuje, nastaví se chyba **res_ErrFIFOunavailable**. Hodnota **OFF** znamená vypnuté FIFO (implicitně). Hodnota **IFAVAIL** znamená zapnutí FIFO, přičemž pokud daný obvod i8250 FIFO neumožňuje, nehlásí se žádná chyba a k obvodu se přistupuje jako k obvodu bez FIFO. Z praktického hlediska se doporučuje spíše používat hodnotu **IFAVAIL** než hodnotu **ON**.

Příklad:

Příklad ukazuje, jak je možné v jednotce COM nastavit parametry komunikace na sudou paritu, přenosovou rychlost 4800 Bd a velikost vstupního vyrovnávacího bufferu na 1000 položek.

```
ChSetParam('NAM=COM PAR=E BD=4800 LRB=1000');
```

Pozn: Všimněte si, že není volána metoda ChSetOneParam, ale metoda ChSetParam.

6.1.2.5. 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**.

6.1.2.6. ChSetBinParam procedura

```
procedure ChSetBinParam(NumName: tChNumName; Code: Word;
                        Param: longint);
```

Metodou **ChSetBinParam** s parametrem NumName příslušným číselnému jménu tohoto komunikačního objektu, lze podle parametru `Code` nastavit tyto nastavení:

`Code = cmd_SetMCR`
: Podle nejnižšího byte parametru Param nastaví obsah Modem Control Registeru - obraz všech výstupních modemových signálů.

`Code = cmd_SetDTR`
: Podle 0.bitu parametru Param nastaví nebo vynuluje modemový signál DTR.

`Code = cmd_SetRTS`
: Podle 0.bitu parametru Param nastaví nebo vynuluje modemový signál RTS.

6.1.2.7. ChGetBinParam funkce

```
function  ChGetBinParam(NumName: tChNumName; Code: Word): longint;
```

Metodou **ChGetBinParam** s parametrem NumName příslušným číselnému jménu tohoto komunikačního objektu, lze podle parametru Code vrátit tyto nastavení:

`Code = cmd_GetMSR`
: Vrátí obsah Modem Status Registeru - obraz všech vstupních modemových signálů.

`Code = cmd_GetDCTS`
: Vrátí v 0.bitu příznak modemového signálu DCTS.

`Code = cmd_GetDDSR`
: Vrátí v 0.bitu příznak modemového signálu DDSR.

`Code = cmd_GetTERI`
: Vrátí v 0.bitu příznak modemového signálu TERI.

`Code = cmd_GetDRLSD`
: Vrátí v 0.bitu příznak modemového signálu DRLSD.

`Code = cmd_GetCTS`
: Vrátí v 0.bitu příznak modemového signálu CTS.

`Code = cmd_GetDSR`
: Vrátí v 0.bitu příznak modemového signálu DSR.

`Code = cmd_GetRI`
: Vrátí v 0.bitu příznak modemového signálu RI.

`Code = cmd_GetRLSD`
: Vrátí v 0.bitu příznak modemového signálu RLSD.

`Code = cmd_GetCntIrq`
: Vrátí počet všech vyvolaných přerušení.

`Code = cmd_GetCntRecB`
: Vrátí počet přerušení způsobených příjmem znaku nebo statusem.

`Code = cmd_GetCntSendB`
: Vrátí počet přerušení způsobených odvysíláním znaku.

`Code = cmd_GetFIFOenabled`
: Vrátí v 0.bitu příznak, je-li skutečně povoleno FIFO (volat po metodě ChOpen).

6.1.2.8. ChOpen procedura

```
procedure ChOpen;
```

Metoda **ChOpen** nastaví technické vybavení komunikačního kanálu, a pokud nastavení proběhlo v pořádku, způsobí přechod kanálu do stavu **CHS_Open**.

6.1.2.9. ChClose procedura

```
procedure ChClose;
```

Metoda **ChClose** uzavře komunikační kanál provedením deinicializace technického vybavení a způsobí přechod do stavu **CHS_Close**. Lze opětovně volat metodu **ChOpen**.

6.1.2.10. ChConnect procedura

```
procedure ChConnect;
```

Před voláním této metody musí být kanál ve stavu **CHS_Open**. Metoda **ChConnect** provede inicializaci pro příjem a vysílání znaků a pokud nastavení proběhlo v pořádku, způsobí přechod do stavu **CHS_Connect**. To znamená, že je možno po daném kanále komunikovat. V tomto stavu je možno přijímat data z komunikační linky, naopak je možno požadovaná data odvysílat.

6.1.2.11. ChDisConnect procedura

```
procedure ChDisConnect;
```

Metoda **ChDisConnect** ukončí navázané komunikační spojení a uvede kanál do stavu **CHS_DisConnect**. Je přerušen příjem a vysílání zpráv. Po volání této metody lze opětovně volat metodu **ChConnect**.

6.1.2.12. ChSendTick procedura

```
procedure ChSendTick;
```

Metoda **ChSendTick** způsobí provedení kroků vysílacích automatů. Je nutné ji periodicky volat během vysílání. **ChSendTick** je rovněž automaticky volána v metodách **ChSendReady** a **ChSend**.

6.1.2.13. ChSend procedura

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

Pokud je kanál ve stavu **CHS_Connect**, způsobí metoda **ChSend** započetí vysílání zprávy délky `Len` uložené na adrese určené ukazatelem `Buff`. Pokud je parametr Len = 0, nebude se vysílat žádná zpráva. Po volání této metody by mělo následovat volání metody **ChSendReady** s testem na **CHS_SendReady** (čekací smyčka do odvysílání zprávy).

6.1.2.14. ChSendReady funkce

```
function  ChSendReady: TChState;
```

Metoda **ChSendReady** způsobí provedení kroku vysílacího automatu na základě volání metody **ChSendTick**. Jako svoji funkční hodnotu vrátí aktuální stav

automatu vysílače komunikačního kanálu, který je uložen v položce **CH_SCtrl**. Pokud kanál není ve stavu **CHS_Connect**, vrací metoda stav **CHS_SendNoReady**.

6.1.2.15. ChSendFlush procedura

```
procedure ChSendFlush;
```

Pokud je kanál ve stavu **CHS_Connect** způsobí metoda **ChSendFlush** ukončení vysílání a přechod automatu vysílače do stavu **CHS_SendReady**.

6.1.2.16. ChReceiveReady funkce

```
function  ChReceiveReady: TChState;
```

Pokud je kanál ve stavu **CHS_Connect** a v přijímacím bufferu jsou přijata nějaká data, vrátí metoda **ChReceiveReady** stav **CHS_ReceiveReady**. V opačném případě vrátí stav **CHS_ReceiveNoReady**.

6.1.2.17. ChReceiveChar funkce

```
function  ChReceiveChar: Byte;
```

Pokud je kanál ve stavu **CHS_Connect** a v přijímacím bufferu jsou přijata nějaká data, navrací metoda **ChReceiveChar** jeden přijatý znak z přijímacího bufferu a nastaví výsledek operace přijímače na status tohoto přijatého znaku. Metodu **ChReceiveChar** je možno volat pouze ve stavu automatu přijímače **CHS_ReceiveReady**, proto před voláním této metody musí předcházet volání metody **ChReceiveReady** s testem na stav **CHS_ReceiveReady**, jinak v případě nepřijetí žádného znaku skončí volání metody **ChReceiveChar** chybou.

6.1.2.18. ChReceive procedura

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

Pokud je kanál ve stavu **CHS_Connect** a je nadefinován buffer pro uložení přijaté zprávy (metodou **ChReceiveBuffer**), provede metoda **ChReceive** přijmutí zprávy a její uložení do přijímacího bufferu. V proměnné `Len` navrací délku přijaté zprávy. Ve svém těle volá metodu **ChReceiveChar** (pro přijetí jednoho znaku zprávy) tak dlouho, dokud je přijímač ve stavu **CHS_ReceiveReady**.

6.1.2.19. ChReceiveFlush procedura

```
procedure ChReceiveFlush;
```

Metoda **ChReceiveFlush** způsobí vyprázdnění přijímacích bufferů a nastaví stav přijímače kanálu **CH_RCtlr** na stabilní stav **CHS_ReceiveNoReady**.

6.2. tAddChnCom

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

6.2.1. Metody

6.2.1.1. ChInit funkce

```
function ChInit: pChnVirt;
```

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

7. Příklad

Příklad ukazuje použití komunikační jednotky ChnCom. Je vytvořen komunikační kanál definovaných vlastností, po kterém je zasílána zpráva a z kterého je poté očekáván příjem zpráv.

```
uses
  uString,
  ChnTypes,
  ChnVirt,
  ChnCom,
  ...

const
  ParamStr : tParamStr =
             'NAM=COM COM=1 IRQ=4 BD=1200 PAR=N BIT=8 STOP=2 LRB=1000';

type
  tMess    = array [0..65500] of Byte;

var
  Chn      : pChnVirt;
  SMess    : ^tMess;
  RMess    : ^tMess;
  LSMess   : Word;
  LRMess   : Word;

begin
  ...
  New(SMess);
  New(RMess);
  ...
  { inicializace Chn }
  Chn:=ChnCollection^.ChNewInit(ChnCom.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;
    { definování místa, kam se má přijatá zpráva uložit }
    ChReceiveBuffer(RMess,SizeOf(tMess));
    if ChReceiveResult<>res_Ok then WriteLn('Chyba');
    ChConnect;
    repeat
      if ChResult<>res_Ok then WriteLn('Chyba');
    until ChReady=CHS_Connect;
    ...
    { naplnění zprávy daty }
```

```
    ...
    { vyslání zprávy }
    if ChSendReady=CHS_SendReady then
    begin
      ChSend(SMess, LSMess);
      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');
    ...
    { ukončení }
    ChDisConnect;
    repeat
      if ChResult<>res_Ok then WriteLn('Chyba');
    until ChReady=CHS_DisConnect;
    ChClose;
    repeat
      if ChResult<>res_Ok then WriteLn('Chyba');
    until ChReady=CHS_Close;
  end;
  { zrušení instance objektu }
  Dispose(Chn,Done);
  ...
end.
```