ChnVMMA

JEDNOTKA DEFINUJÍCÍ KOMUNIKAČNÍ PROTOKOL SPOLEČNOSTI "THE EUROPEAN VENDING MACHINE MANUFACTURERS ASSOC."

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: 07.04.2004
Datum posledního uložení dokumentu: 07.04.2004
(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	1.0X	Wil	02.06.2003	První vydání.
1.10	1.XX	Wil	07.04.2004	Změna číslování chybových konstant res_ErrXxx dle standardu.

1.2. Účel dokumentu

Tento dokument slouží jako popis jednotky definující komunikační protokol „The Internal Communication Protocol“ (I.C.P.) verze 2 společnosti „The European Vending Machine Manufacturers Association“ pro komunikaci s jejich zařízeními.

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, dále s dokumentem „ChnTypes“ definujícím konstanty a typy pro sériovou komunikaci a dokumentem „ChnComPB“ definujícím knihovnu pro fyzický přenos dat po RS232/485 s využitím rozlišení datových a adresních bytů dle paritního bitu.

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“.

Master - Jednotka nebo řídicí systém komunikující protokolem implementovaným v knihovně ChnVMMA, které pomocí této komunikace ovládá zařízení Slave.

Slave - Zařízení (též periferie) firmy „The European Vending Machine Manufacturers Association“, které odpovídá na zprávy Master stanice a vykonává příslušné příkazy.

3. Úvod

Knihovna definuje formát komunikačního protokolu používaného při komunikaci se zařízeními firmy The European Vending Machine Manufacturers Association. Knihovna z tohoto komunikačního protokolu implementuje stranu Master pro komunikaci s mincovníkem. Obstarává zabezpečení dat, vkládání a vyjímání nadbytečností, tak jak to tento protokol předepisuje. Fyzický přenos dat je zajištěn prostřednictvím nižší komunikační vrstvy, přičemž tento protokol vyžaduje jako fyzickou vrstvu knihovnu ChnComPB.

Knihovna ChnVMMA definuje komunikační objekt **tChnVMMA**, který je dědicem od rodičovského komunikačního objektu tChnVirt. Instance objektu tChnVMMA 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.

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

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

4. Konstanty a typy

```
cVerNo = např. $0102; { BCD formát }
cVer   = např. '01.02,17.03.2003';
```

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

```
cName = 'VMMA';
```

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

4.1. Konstanty chybových kódů

Následující chyby mohou vracet metody ChReceiveResult, ChSendResult a případně i ChResult.

```
Res_ErrFrame    = $20;
```

Chybný formát zprávy - přijatou zprávu nelze akceptovat.

```
res_ErrSum      = $21;
```

Chyba kontrolního součtu SUM8 - přijatou zprávu nelze akceptovat.

```
res_ErrLen      = $22;
```

Chyba délky zprávy - přijatou zprávu nelze akceptovat.

4.2. Znaky jednobytových zpráv protokolu

```
ACK = $00;
NAK = $FF;
RET = $AA;
```

Konstanta **ACK** definuje kód znaku zprávy od Slave stanice, která tím signalizuje krátké pozitivní potvrzení přijetí a vykonání příkazu od Master stanice.

Konstanta **NAK** definuje kód znaku zprávy od Slave stanice, která tím signalizuje krátké negativní potvrzení o nepřijetí příkazu od Master stanice.

Konstanta **RET** definuje kód znaku zprávy od Slave stanice, která si tím vyžádá opakování zprávy od Master stanice.

4.3. Adresy periferií

```
cDNode_Changer     = $08;
```

Konstanta definuje adresu mincovníku.

```
cDNode_CardReader  = $10;
```

Konstanta definuje adresu čtečky karet.

```
cDNode_AuditSystem = $18;
```

Konstanta definuje adresu systému pro vyúčtování.

```
cDNode_Display     = $20;
```

Konstanta definuje adresu displeje.

```
cDNode_EnManagSys  = $28;
```

Konstanta definuje adresu systému pro řízení energetických zdrojů.

```
cDNode_BillValider = $30;
```

Konstanta definuje adresu kontrolovače účtenek.

```
cDNode_VendMachin1 = $F0;
```

Konstanta definuje adresu specifické periferie 1.

```
cDNode_VendMachin2 = $F8;
```

Konstanta definuje adresu specifické periferie 2.

Tyto konstanty se používají pro metodu **ChDestNode** nebo **ChSetParam** komunikačního objektu pro určení periferie, se kterou se bude právě komunikovat.

4.4. Konstanty hlavních příkazů pro všechny periferie

```
CM_Reset       = $00;
```

Konstanta definuje kód příkazu pro provedení resetu periferie.

```
CM_Status      = $01;
```

Konstanta definuje kód příkazu pro čtení statusu nastavení periferie.

```
CM_Poll        = $03;
```

Konstanta definuje kód příkazu pro čtení stavu aktivity.

```
CM_Extended    = $07;
```

Konstanta definuje kód příkazu pro další Sub-příkazy.

4.5. Konstanty hlavních příkazů pro mincovník

`CM_TubeStatus = $02;`
Konstanta definuje kód příkazu pro čtení stavu zásobníku mincí.

`CM_CoinType = $04;`
Konstanta definuje kód příkazu pro nastavení povolení výdeje mincí.

`CM_Dispense = $05;`
Konstanta definuje kód příkazu pro provedení výdeje mincí.

4.6. Konstanty Sub-příkazů mincovníku

`SC_Ident = $00;`
Konstanta definuje kód Sub-příkazu pro čtení identifikace zařízení.

`SC_EnbFeature = $01;`
Konstanta definuje kód Sub-příkazu pro nastavení volitelných voleb.

`SC_AltPayout1 = $02;`
Nespecifikovaný Sub-příkaz.

`SC_AltPayout2 = $03;`
Nespecifikovaný Sub-příkaz.

`SC_Diagnostic = $FF;`
Nespecifikovaný Sub-příkaz.

4.7. Konstanty stavů aktivity mincovníku

`pl_EscrowReq = $01;`
Konstanta definuje stav detekce otevření krytky.

`pl_PayoutBusy = $02;`
Konstanta definuje stav, že mincovník je zaneprázdněn.

`pl_NoCredit = $03;`
Konstanta definuje stav, že mince byla přijata, ale nedostala se na správné místo.

`pl_TubeSensDef = $04;`
Konstanta definuje defektní stav nekterého ze senzorů mincovních zasobníků.

`pl_DoubleArriv = $05;`
Konstanta definuje stav, že byly vhozeny dvě mince příliš rychle po sobe a bez jejich detekce.

`pl_AcceptUnplug= $06;`
Konstanta definuje stav, že mincovní přijímač byl odstranen.

`pl_TubeJam = $07;`
Konstanta definuje stav zaseknutí mincovního zasobníku.

`pl_ROMsumErr = $08;`
Konstanta definuje chybový stav interního kontrolního součtu.

`pl_CoinRoutErr = $09;`
Konstanta definuje stav, že mince byla přijata, ale nepropadla dále.

```
pl_ChangerBusy = $0A;
```
Konstanta definuje stav, že mincovník je zaneprázdněn a nemůže detailně odpovídat na příkazy.

```
pl_ChangerReset= $0B;
```
Konstanta definuje stav, že mincovník byl resetován.

```
pl_CoinJam     = $0C;
```
Konstanta definuje stav, že se mince zasekla.

4.8. Definice typů mincí

```
MaxCoinType = 15;
```
Konstanta definující maximální počet typů mincí mínus 1, tj. 16.

```
tCoins = 0..MaxCoinType;
```
Intervalovy typ označení mincí.

4.9. Typy zpráv

```
tMessType =
  (tpAckNakRet, { jednoduchá zpráva ACK, NAK nebo RET }
   tpDataMess); { datová/příkazová zpr8va }
```

4.10. Struktury přijímacích a vysílacích bufferů

```
pMaSendRecord = ^tMaSendRecord;
tMaSendRecord = record
    case MessType         : tMessType of
      tpAckNakRet:
        (AckNakRet        : byte);
      tpDataMess:
        (case Code        : byte of
          CM_Reset      ,
          CM_Status     ,
          CM_TubeStatus ,
          CM_Poll       :
            ();
          CM_CoinType   :
            (EnbCoins     : word;
             EnbManDispens: word;
            );
          CM_Dispense   :
            (CoinType     : tCoins;
             CoinNumber   : tCoins;
            );
          CM_Extended   :
            (case SubCode : byte of
            );
        );
end;
```

TMaSendRecord je typ variantního záznamu, který svou strukturou odpovídá datům protokolu posílaným Master stanicí ven z jednotky, přičemž je zbaven nadbytečností, které jsou při vysílání doplněny. Položka **MessType** definuje základní typ vysílané zprávy.

V případě krátké zprávy tpAckNakRet se používá položka **AckNakRet**, která obsahuje některou z konstant ACK, NAK nebo RET.

V případě dlouhé datové zprávy tpDataMess se používá položka **Code**, která definuje kód zprávy (viz konstanty CM_Xxx). U datové zprávy pro nastavení povolení výdeje mincí (Code = CM_CoinType) se používají položky **EnbCoins** (bitové povolení přijímání jednotlivých mincí) a **EnbManDispens** (bitové povolení manuálního vracení jednotlivých mincí). U datové zprávy pro provedení výdeje mincí (Code = CM_Dispense) se používají položky **CoinType** (typ vydávaných mincí) a **CoinNumber** (počet vydávaných mincí). U datové zprávy pro Sub-příkazy (Code = CM_Extended) se používá položka **SubCode** definující kód Sub-příkazu (viz konstanty SC_Xxx).

```
tPollType =
  (poll_UnUsed,    { nepoužito }
   poll_AckNak,    { ACK nebo NAK }
   poll_Dispensed, { Coins Dispensed Manually }
   poll_Deposited, { Coins Deposited }
   poll_Status,    { Status }
   poll_Slug);     { propadnutí mince }
```

Výčtový typ **tPollType** definuje druhy odpovědí od Slave stanice na zprávu **Poll** od Master stanice. Zprávou Poll od Master stanice (položky ve vysílacím bufferu typu tMaSendRecord: MessType = tpDataMess, Code = CM_Poll) se provádí dotaz na stav aktivity dotazované periferie. Touto zprávou by se měla Master stanice dotazovat dané periferie s periodou 25 až 200ms. Pokud daná periferie přestane odpovídat (timeout pro mincovník je 2s), může jí Master stanice posílat zprávu Poll s periodou 10s. Zprávu Poll by měla také Master stanice vyslat periferii v případě, že ta odpoví zprávou NAK nebo RET.

```
tRecPollRecord = record
    case PollType    : tPollType of
      poll_UnUsed   :
        ();
      poll_AckNak   :
        (PollAckNak   : byte);
      poll_Dispensed:
        (CoinDisType  : tCoins;
         CoinDisNumber: 0..7;
         CoinsInTube  : byte;
        );
      poll_Deposited:
        (CoinTypeDep  : tCoins;
         CoinRouting  : byte;
         CoinsAccept  : byte;
        );
      poll_Status:
        (Status       : byte;
        );
      poll_Slug:
        (Slug         : byte;
        );
end;
```

TRecPollRecord je typ variantního záznamu, který svou strukturou odpovídá datům protokolu přijímaným Master stanicí na zprávu Poll (viz struktura přijímacího bufferu tMaRecRecord níže), přičemž je zbaven nadbytečností, které jsou při příjmu odstraněny.

Pokud mincovník neprovádí žádnou činnost, vrátí na zprávu Poll ACK nebo NAK, tj. **PollType = poll_AckNak** a v položce **PollAckNak** bude kód ACK nebo NAK.

Pokud mincovník provádí výdej mincí, vrátí na zprávu Poll zprávu **PollType = poll_Dispensed** a následující položky: **CoinDisType** definuje typ vydávaných mincí, **CoinDisNumber** definuje počet vydávaných mincí a **CoinsInTube** definuje počet zbývajících mincí v zásobníku.
Pokud mincovník přijal nějaké mince, vrátí na zprávu Poll zprávu **PollType = poll_Deposited** a následující položky: **CoinTypeDep** definuje typ vložené mince, **CoinRouting** definuje místo, kam byla přijatá mince umístěna (viz konstanty cr_Xxx), a **CoinsAccept** definuje počet mincí v zásobníku po přijetí.
Pokud mincovník rozpoznal nějakou důležitou událost, vrátí na zprávu Poll zprávu **PollType = poll_Status** a v položce **Status** kód události (viz konstanty pl_Xxx).

```
pMaRecRecord = ^tMaRecRecord;
tMaRecRecord = record
    case Code             : byte of
      CM_Reset      ,
      CM_CoinType   ,
      CM_Dispense   :
        (AckNak           : byte);
      CM_Status     :
        (Level            : 2..3;
         Country          : word;
         CoinTypeRouting  : word;
         CoinScalFact     : byte;
         DecimalPlac      : byte;
         CoinCredit       : array[tCoins]of byte;
        );
      CM_TubeStatus :
        (Dummy1_1         : byte;
         TubeFull         : word;
         TubeCoins        : array[tCoins]of byte;
        );
      CM_Poll       :
        (Dummy1_3         : array[1..3]of byte;
         RecPollArr       : array[1..16]of tRecPollRecord;
        );
      CM_Extended   :
        (case SubCode : byte of
           SC_Ident      :
             (ManufCode : string[3];
              SerialNum : string[12];
              ModelTun  : string[12];
              SWVer     : string[2];
              OptFeatur : longint;
             );
        );
end;
```

TMaRecRecord je typ variantního záznamu, který svou strukturou odpovídá datům protokolu přijímaným Master stanicí, přičemž je zbaven nadbytečností, které jsou při příjmu odstraněny. Položka **Code** definuje kód zprávy od Master stanice, na kterou Slave odpovídá.
Na zprávy **Reset, CoinType** a **Dispense** odpovídá Slave zprávou ACK nebo NAK, tj. používá se položka **AckNak**, která obsahuje kód ACK nebo NAK.

Při odpovědi na zprávu **Status** se používají položky: **Level** - úroveň ovládání mincovníku, **Country** - kód země, **CoinTypeRouting** - bitové určení typů mincí ukládaných do zásobníku, **CoinScalFact** - nejmenší mince v nejmenších měnových jednotkách (např. 100 pro 1Kč nebo 5 pro USA nickel), **DecimalPlac** - počet desetinných míst (např. 2), **CoinCredit** - hodnota jednotlivých typů mincí v nejmenších měnových jednotkách (např. 100, 200 pro 1Kč, 2Kč).

Při odpovědi na zprávu **TubeStatus** se používají položky: **TubeFull** - bitové příznaky určující plnost mincovních zásobníků, **TubeCoins** - počty mincí v jednotlivých mincovních zásobnících. Položka **Dummy1_1** je pouze pro zarovnání na sudou adresu.

Při odpovědi na zprávu **Poll** se používá pole **RecPollArr**, kde jedna položka tohoto pole je typu tRecPollRecord, který byl popsán výše. Položka **Dummy1_3** je pouze pro zarovnání na sudou adresu.

Při odpovědi na rozšířenou zprávu (mincovník level 3) se používá položka **SubCode**, která určuje kód sub-příkazu od Master stanice. Je-li **SubCode = SC_Ident**, používají se následujícíc položky: **ManufCode** - výrobní kód, **SerialNum** -: továrně přiřazené sériové číslo, **ModelTun** - továrně přiřazené ladicí číslo a číslo modelu, **SWVer** - verze SW, **OptFeatur** - bitově volitelná nastavení.

```
pSlSendRecord = ^tSlSendRecord;
tSlSendRecord = tMaRecRecord;
```

TSlSendRecord je typ variantního záznamu, který svou strukturou odpovídá datům protokolu vysílaným Slave stanicí, přičemž je zbaven nadbytečností, které jsou při vysílání doplněny. Svou vnitřní strukturou je shodný se záznamem pro přijímací buffer Master stanice.

```
pSlRecRecord  = ^tSlRecRecord;
tSlRecRecord  = tMaSendRecord;
```

TSlRecRecord je typ variantního záznamu, který svou strukturou odpovídá datům protokolu přijímaným Slave stanicí, přičemž je zbaven nadbytečností, které jsou při příjmu odstraněny. Svou vnitřní strukturou je shodný se záznamem pro vysílací buffer Master stanice.

5. Objekty

5.1. tChnVMMA

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_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 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_IndexB  := 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 tChnFesto 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 některé z hodnot konstant cDNode_Xxx. Pokud je stanice označena jako Master, nemá tento parametr smysl.

DNO=DNode

Parametrem **DNO** ("Destination Node") se určuje číslo (adresa) stanice **CH_DNode** v komunikační síti, které budou zprávy určeny. Tuto položku je také možno definovat prostřednictvím metody **ChDestNode**. DNode může nabývat některé z hodnot konstant cDNode_Xxx. Pokud je stanice označena jako Slave, nemá tento parametr smysl.

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/tSlSendRecord, na který ukazuje parametr Buff. Parametr Len udává délku vysílacího bufferu pro vysílání. Tento parametr není nutno správně zadat, jelikož protokol si umí délku vysílané zprávy spočítat sám na základě nastavených položek záznamu.

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 přijetí 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 tMaRecRecord/tSlRecRecord, úvodní a zakončovací řídicí znaky ze zprávy metoda vyhodnotí a pro uživatele odstraní. V parametru Len vrátí velikost přijaté zprávy. 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. 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. tAddChnVMMA

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

6. Příklad

Následující příklad ukazuje způsob vyslání zprávy pro výdej dvou mincí typu 0 z mincovníku. Fyzický přenos se realizuje prostřednictvím knihovny ChnComPB.

```
uses
  NumToStr,
  uString,
  ChnVirt,
  ChnComPB,
  ChnVMMA;
```

```
const
  ParamStr : tParamStr =
             'NAM=FESTO MAS=MASTER LSB=500 DNO=$08 ' +
             'NAM=COMPB COM=1 IRQ=4 BD=9600 BIT=8 STOP=1 ' +
             'LRB=1000 PB=ON';

var
  Chn   : pChnVirt;
  SMess : pMaSendRecord;
  RMess : pMaRecRecord;
  LSMess: word;
  LRMess: word;

begin
  ...
  New(SMess);
  New(RMess);
  ...
  { vytvoření instance Chn }
  Chn:=ChnCollection^.ChNewInit(ChnVMMA.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 (např. výdej dvou mincí typu 0)}
    with SMess^ do
    begin
      MessType  := tpDataMess;
      Code      := CM_Dispense;
      CoinType  := 0;
      CoinNumber:= 2;
    end;
    { vyslání zprávy }
    if ChSendReady=CHS_SendReady then
    begin
      ChSend(SMess, 0);
      { 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 výdej mincí)}
      with RMess^ do
      begin
        if (Code   = CM_Dispense) and
           (AckNak = ACK) then
          writeln('Ok byly vydány 2 mince typu 0')
        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.
```