UDPPrt

IMPLEMENTACE PROTOKOLU PRT(DF0) PŘENÁŠENÉHO POMOCÍ UDP

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 1.12

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: 08.04.2004
Datum posledního uložení dokumentu: 08.04.2004
(Datum vydání a posledního uložení dokumentu musí být stejné)

Obsah :

1. O dokumentu

1.1. Revize dokumentu

Verze dokumentu	Verze SW	Autor	Datum vydání	Popis změn
1.00	1.XX	Hv	8.1.2003	První vydání.
1.01	1.XX	Hv	19.3.2003	Zpřesnění popisu objektu a jeho metod.
1.10	1.XX	Hv	20.5.2003	Úprava dokumentu dle ISO9000.
1.11	1.XX	Hv	19.08.2003	Úprava interface objektu z důvodu přechodu na rozšířenou standardní knihovnu Crc16. Dřívější verze používaly ExCrc16.
1.12	1.XX	Wil	08.04.2004	Grafická úprava dokumentu. Opravy popisu v kapitolách 1.x.

1.2. Účel dokumentu

Tento dokument slouží jako popis komunikační knihovny UDPPrt pro přenos dat pomocí firemního protokolu typu DF0 pomocí UDP na síti Ethernet.

Pozn: Protokol typu DF0 se standardně používá ve firmě SofCon v celé řadě aplikací pro přenos dat po sériové komunikační lince RS232, RS485, modem apod. K tomuto účelu slouží knihovna ChnPrt definující protokol DF0 a dále některá z knihoven nižší fyzické vrstvy (např. ChnCom). Tj. knihovna UDPPrt na síti Ethernet je obdobou knihovny ChnPrt na sériové lince COM.

1.3. Rozsah platnosti

Určen pro programátory a uživatele programového vybavení SofCon.

1.4. Související dokumenty

Pro čtení tohoto dokumentu je nutné se orientovat v koncepci komunikačních objektů, popsána v manuálu ChnVirt. Na tomto konceptu je vytvořen objekt tUDPPrt popisovaný v tomto manuálu. Tento objekt je navázán na vrstvu UDP popsanou v manuálu „CoUDP“, odvozenou od CoBase.

Implementovaný protokol (DF0) je popsán v dokumentu „ChnSof“ nebo „ChnPrt“.

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

Hlavním úkolem této knihovny je vytvořit komunikační vrstvu kompatibilní se současnými aplikacemi a umožňující změnou parametrizačního řetězce přenos dat po sítích Ethernet.

K tomuto účelu knihovna definuje komunikační objekt tUDPPrt, který je potomkem objektu tChnVirt. Instance tohoto objektu představuje vyšší komunikační vrstvu, jejímž úkolem je přenos dat pomocí dále definovaného rámce. Úkolem spodních vrstev je tato data přenést po komunikační lince, tzn. provést fyzický přenos.

Přestože objekt tUDPPrt vychází z objektu tChnPrt, nelze u něj zvolit spodní vrstvu komunikačních objektů pomocí parametrizační řetězce, protože je přímo navázán na objekt tCoUDP odvozený od tCoBase. Objekt tCoUDP implementuje komunikační kanál po síti Ethernet. Tento kanál je složen z několika vrstev. Podrobný popis těchto vrstev naleznete v samostatných manuálech CoBase, CoIPv4 a CoUDP. V rámci zjednodušení popisu činnosti objektu tUDPPrt budou tyto vrstvy dále označovány jako

síťová vrstva	v této vrstvě jsou především ovladače fyzického zařízení, např. síťové karty.
IP vrstva	v této vrstvě jsou implementovány některé služby a zpracování hlaviček související s přenosem dat.
UDP vrstva	v této vrstvě jsou implementovány služby související s přenosem datagramu.

Po komunikačním kanálu je přenášen stejný rámec jako v knihovně ChnPrt. Přestože jeho součástí je adresa odesílatele a příjemce (dále adresáta), umožňuje poslat zprávu všem připojeným stanicím najednou (dále broadcast). Dalšími vlastnosti tohoto rámce je zajištění zabezpečení dat, vkládání a vyjímání nadbytečnosti.

V knihovně je dále implementován objekt tAddUDPPrt, který je potomkem objektu tAddChnVirt. Úkolem tohoto objektu je, aby po přilinkování této jednotky do aplikace (příkazem „**uses UDPPrt**“), objekt tUDPPrt zařadil do seznamu správce komunikačních objektů. Později při zpracování parametrizačního řetězce tento správce zajistí automatické vytvoření a propojení instance tohoto objektu s dalšími komunikačními knihovnami. Takto propojené instance objektů se potom nazývají řetězec komunikačních objektů.

Protože objekt tUDPPrt je dědicem objektu tChnVirt, jsou v této příručce popsány pouze předefinované metody a některé vlastnosti typické pro tuto komunikaci případně objekt. Nepopsané metody a konstanty najdete v samostatných manuálech používaných jednotek nebo rodičovských objektů.

4. Popis rámce

Všechny datové přenosy se uskutečňují pomocí rámců. Jejich struktura zahrnuje zabezpečení dat pomocí CRC16. V tabulce nebude zahrnuta transparence, která se řeší přidáním DLE a bude popsána dále.

	Struktura rámce						
Délka (B)	1	1	1	2	LEN	2	1
Obsah	SOH	DNODE	NODE	LEN	DATA (protokol)	CRC	ETX

DLE znak transparence, 010h
SOH začátek zprávy, 001h
DNODE adresa příjemce v síti <0;255>
NODE adresa odesílatele v síti <0;255>
LEN délka datové zprávy DATA v bytech<0, 32734> (bez znaků DLE)
DATA datová zpráva
CRC zabezpečení dat – je generován z SOH, DNODE, NODE, LEN a DATA (bez znaků DLE); CRC16 = $x^{16}+ x^{15} +x^2 + 1$
ETX konec zprávy, 003h

Při vysílání zprávy jsou jednotlivé znaky přenášeny za použití následujících pravidel:

- SOH se přenáší jako dvojice DLE, SOH – 010h,001h
- ETX se přenáší jako dvojice DLE, ETX – 010h,003h
- ostatní znaky zbylých částí zprávy (DNODE, NODE, LEN, DATA a CRC) se přenáší následovně:
  - znak se *nerovná DLE*, přenáší se jako jeden znak,
  - znak se *rovná DLE*, přenáší se jako dvojice DLE, DLE – 010h,010h.

Po aplikaci předchozích pravidel lze v proudu dat jednoznačně určit začátek a konec rámce. Začátek rámce se určí příjmem dvojice znaků DLE, SOH a konec rámce se určí příjmem dvojice znaků DLE, ETX.

Velikost přenášených dat v poli DATA je omezena na 32734 Byte.

V současné implementaci musí rámec vždy začínat na začátku přijatých dat UDP paketu a musí být pouze v jednom paketu, který může být rozdělen do více fragmentů. Ale to již zajišťuje knihovna CoUDP popsaná v samostatném manuálu. Pokud rámec nezačíná na této pozici nebo je rozdělen do více UDP paketů, nejsou tyto rámce vyhodnoceny jako platné a jsou ignorovány tzn. nastaví se chybový kód a pokračuje se v příjmu paketů.

5. Činnost komunikačního objektu

V další části této kapitoly budou popsány jednotlivé stavy, kterými prochází jednotlivé automaty objektu tUDPPrt. V diagramech bude použita následující symbolika:

- Silnými čarami budou znázorněny přechody vyvolané metodami nad touto šipkou. Poněvadž tyto metody mohou způsobit operace trvající delší dobu případně skončit chybou, měla by se vždy provádět kontrola chyb a tikání automatu pomocí dále uvedených metod.
- Slabými čarami budou znázorněny přechody po dosažení požadovaných podmínek.
- Kroužek se silnou čarou znamená stabilní stav.
- Kroužek se slabou čarou znamená nestabilní stav tj. stav, kterým se prochází po určitou dobu než jsou splněny požadované podmínky pro přechod do stabilního stavu.

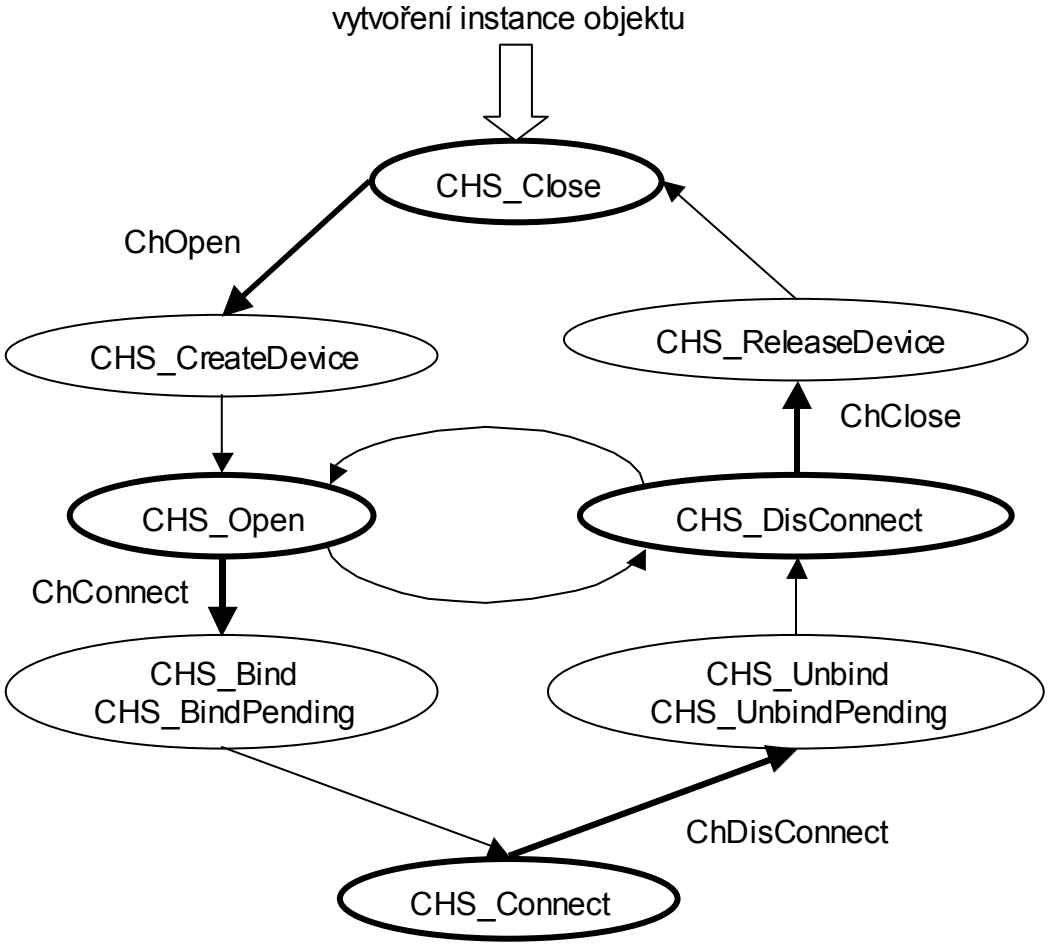
Pokud je metoda zavolána v neplatném stavu, automat zůstává ve stejném stavu a vrací se chyba.

5.1. Stavy komunikačního kanálu

Na následujícím obrázku jsou znázorněny stavy kanálu, které vrací metoda ChReady. Stabilní stavy vrací metody ChState. Interně tyto funkce volají metodu ChTick, která při splnění požadovaných podmínek provede přechod do dalšího stavu tzn. tiká automatem komunikačního kanálu. V opačném případě automat zůstane ve stejném stavu.

Dále budou popsány stavy komunikačního kanálu:

CHS_Close	objekty spodních vrstev neexistují a lze nastavit všechny jejich vlastnosti. V tomto stavu nelze vysílat ani přijímat data.
CHS_Open, CHS_DisConnect	objekty spodních vrstev jsou vytvořeny a lze nastavit některé jejich vlastnosti. V tomto stavu nelze vysílat ani přijímat data.
CHS_Connect	objekty spodních vrstev jsou vytvořeny a lze nastavit některé jejich vlastnosti. V tomto stavu lze jak vysílat tak přijímat data.
CHS_CreateDevice, CHS_ReleaseDevice, CHS_Bind, CHS_BindPending, CHS_Unbind, CHS_UnbindPending	nestabilní stavy, při kterých se vytváří nebo ukončuje komunikační kanál případně spojuje nebo navazuje spojení. Tyto stavy trvají po dobu než se splní požadované podmínky pro přechod do dalšího stavu.

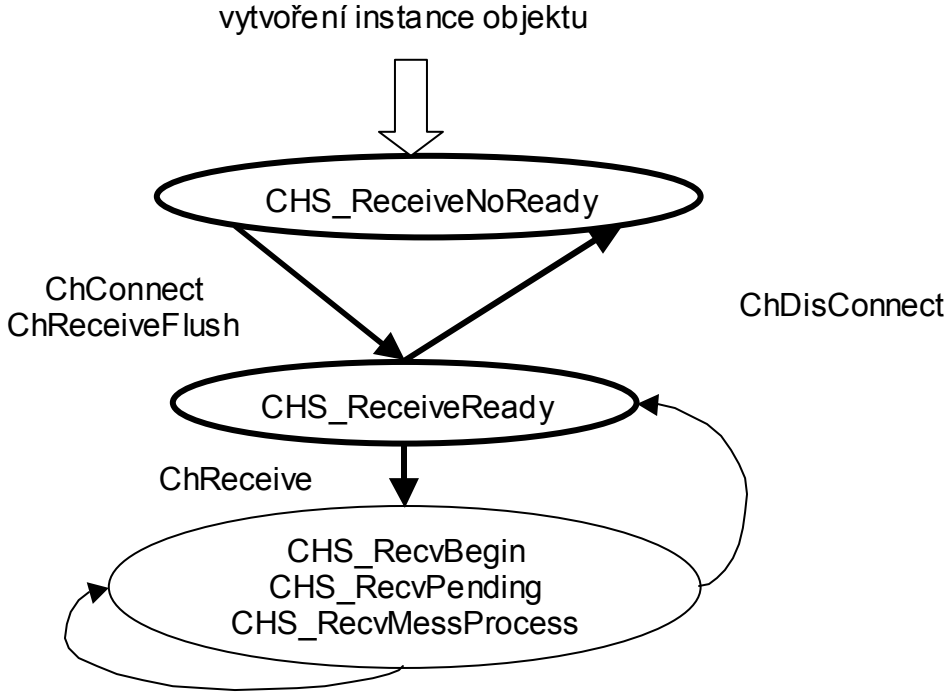


5.2. Stavy přijímacího automatu

Na následujícím obrázku jsou znázorněny stavy přijímacího automatu, které vrací metoda ChReceiveReady. Interně tato funkce volá metodu ChReceiveTick, která při splnění požadovaných podmínek provede přechod do dalšího stavu tzn. tiká přijímacím automatem. V opačném případě automat zůstane ve stejném stavu.

Dále budou popsány stavy přijímacího automatu:

CHS_ReceiveNoReady	v tomto stavu je kanál buď ve stavu různém od CHS_Connect nebo není žádná zpráva přijata
CHS_ReceiveReady	v tomto stavu je kanál ve stavu CHS_Connect a je přijata nějaká platná zpráva – CRC a hlavička byla bez chyby.
CHS_RecvBegin, CHS_RecvPending, CHS_RecvMessProcess	v tomto stavu je kanál ve stavu CHS_Connect a je přijímána nějaká zpráva.

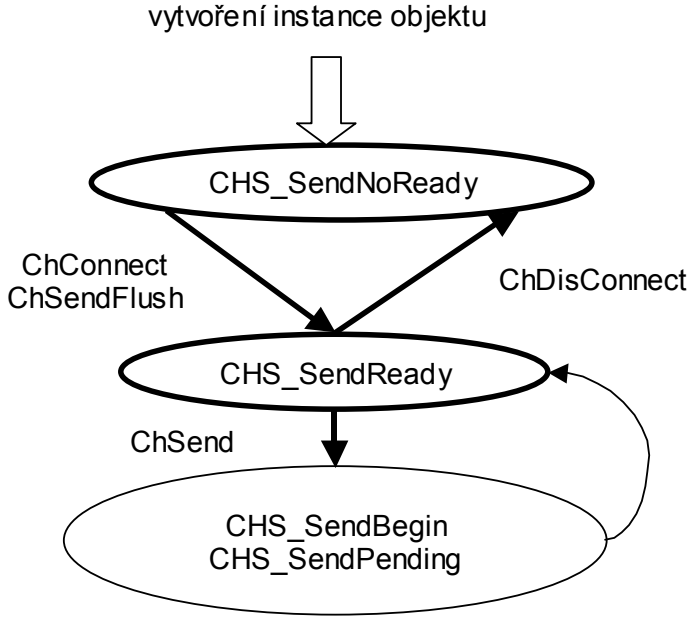


5.3. Stavy vysílacího automatu

Na následujícím obrázku jsou znázorněny stavy vysílacího automatu, které vrací metoda ChSendReady. Interně tato funkce volá metodu ChSendTick, která při splnění požadovaných podmínek provede přechod do dalšího stavu tzn. tiká vysílacím automatem. V opačném případě automat zůstane ve stejném stavu.

Dále budou popsány stavy vysílacího automatu:

CHS_SendNoReady	v tomto stavu je kanál ve stavu různém od CHS_Connect a není připraven vysílat žádné zprávy
CHS_SendReady	v tomto stavu je kanál ve stavu CHS_Connect a je připraven odvysílat zadanou zprávu
CHS_SendBegin, CHS_SendPending	v tomto stavu je kanál ve stavu CHS_Connect a vysílá zadanou zprávu



6. Konstanty

```
cName  = 'UDPPRT';
```

konstanta obsahuje název jednotky a zároveň určuje jméno objektu v parametrizačním řetězci, pomocí kterého lze nastavit parametry této jednotky.

```
cVerNo = např. $0102;
cVer   = např. '01.02,17.03.2003';
```

konstanty udávají verzi a poslední změnu jednotky ve standardním formátu definovaném v knihovně LibVer.

6.1. Chybové kódy

`Res_ErrSocket = $00B0;`
Tato chyba se vrací, pokud se při otevírání komunikačního kanálu metodou ChOpen nepodaří otevřít požadovaný socket. Popis socketu najdete v samostatném manuálu CoBase, CoUDP a CoINET.

`Res_ErrStack = $00B1;`
Tato chyba se vrací, pokud se při otevírání komunikačního kanálu metodou ChOpen nepodaří otevřít požadovaný zásobník zařízení. Popis zásobníku zařízení najdete v samostatném manuálu CoBase a CoINET.

`Res_ErrBind = $00B2;`
Tato chyba se vrací, pokud se při navázání spojení metodou ChConnect nepodaří navázat spojení s otevřeným socketem. Implicitně tato knihovna přijímá pakety od všech jednotek, které komunikují na zadaném portu. Popis navazování spojení najdete v samostatném manuálu CoBase a CoUDP.

`Res_ErrUnbind = $00B3;`
Tato chyba se vrací, pokud se při rozpojování spojení metodou ChDisConnect nepodaří uvolnit spojení s používaným socketem. Popis rozpojování spojení najdete v samostatném manuálu CoBase a CoUDP.

`Res_ErrBufferSize = $00B4;`
Tato chyba se vrací, pokud se při zavolání metody ChSend zjistí, že vysílaná zpráva (holá data) je větší než buffer pro vysílání.

`Res_ErrMsgSize = $00B5;`
Tato chyba se vrací, pokud se při zavolání metody ChSend zjistí, že rámec (vysílaná data s hlavičkou, CRC atd.) je větší než buffer pro vysílání.

`Res_ErrRecvBuffer = $00B6;`
Tato chyba se vrací, pokud při příjmu zprávy metodou CoRecvBufferFrom došlo k chybě. Popis chyb metody CoRecvBufferFrom najdete v samostatném manuálu CoUDP.

`Res_ErrSendBuffer = $00B7;`
Tato chyba se vrací, pokud při vysílání zprávy metodou CoSendBufferTo došlo k chybě. Popis chyb metody CoSendBufferTo najdete v samostatném manuálu CoUDP.

`Res_ErrCrc = $20;`
Tato chyba se vrací, pokud se při zpracování přijatého rámce zjistí, že CRC přijatého rámce nesouhlasí s hodnotou CRC obsaženou v rámci.

`Res_ErrSOH = $21;`
Tato chyba se vrací, pokud se při zpracování přijatého rámce zjistí, že obsahuje druhý znak SOH.

`Res_ErrETX = $22;`
Tato chyba se vrací, pokud se při zpracování přijatého rámce zjistí, že znak ETX je na nepovoleném místě.

`Res_ErrLen = $23;`
Tato chyba se vrací, pokud se při zpracování přijatého rámce zjistí, že velikost přijaté zprávy v rámci překračuje velikost buffer pro příjem zprávy.

Dále popisované konstanty jsou definovány v jednotce ChnVirt. Přesný popis těchto konstant je uveden v manuálu ChnVirt. Zde je uveden jejich stručný popis pro úplnost chybových kódů.

`Res_Ok = $0000;`
Volaná funkce proběhla bez chyby.

`Res_ErrNoReceiveReady = $00c0;`
Tato chyba se vrací, pokud se při volání metody ChReceive zjistilo, že probíhá příjem zprávy.

`Res_ErrNoClose = $00e0;`
Tato chyba se vrací, pokud se při volání metody ChOpen zjistilo, že komunikační kanál není zavřen viz CHS_Close.

`Res_ErrNoOpen = $00e1;`
Tato chyba se vrací, pokud se při volání metody ChConnect zjistilo, že komunikační kanál není otevřen viz CHS_Open.

`Res_ErrNoConnect = $00e2;`
Tato chyba se vrací, pokud se při volání metod ChSend, ChReceive, ChSendFlush a ChReceiveFlush zjistí, že komunikační kanál není spojen CHS_Connect.

`Res_ErrOpen = $00e3;`
Tato chyba se vrací, pokud se při volání metody ChOpen zjistí, že nejsou alokovány buffery pro příjem a vysílání zprávy.

`Res_ErrConnect = $00e4;`
Tato chyba se vrací, pokud se při navazování spojení metodou ChConnect zjistí, že nejsou alokovány buffery pro příjem a vysílání zpráv.

`Res_ErrChannelNoExist = $00fb;`
Tato chyba se vrací, pokud se při volání metody ChClose, ChConnect, ChDisConnect, ChSend, ChReceive, ChSendFlush a ChReceiveFlush zjistí, že neexistuje otevřený socket.

`Res_ErrParamStr = $00fc;`
Chyba v parametrizačním řetězci případně pokus o nastavení parametru, který smí být nastaven pouze v rozpojeném stavu. Popis jednotlivých parametrů naleznete v samostatných manuálech komunikačních knihoven – ChnVirt, CoBase, CoUDP atd.

`Res_Err = $00ff;`
Nespecifikovaná chyba zpravidla při dekódování přijaté zprávy.

6.2. Řídící znaky protokolu

`SOH = $01;`
Definuje kód znaku, který jednoznačně identifikuje začátek rámce.

`DLE = $10;`
Definuje kód znaku, který slouží pro rozlišení řídících znaků od datových znaků.

`ETX = $03;`
Definuje kód znaku, který jednoznačně identifikuje konec rámce.

7. Typy struktur

Dále popisované typy jsou definovány v uvedených jednotkách. Zde je uveden jejich stručný popis pro úplnost používaných typů.

```
PIpAddress = ^TIpAddress;
TIpAddress = Longint;
```

Deklarace typu popisující IP adresu. Bližší popis lze najít v manuálu CoIPv4.

```
PUdpAddress = ^TUdpAddress;
TUdpAddress = packed record
  Size    : Byte;
  Address : TIpAddress;
  Port    : Word;
end;
```

Deklarace struktury popisující adresu pro protokol UDP. Bližší popis lze najít v manuálu CoUDP.

```
TCoStatus = Word;
```

Deklarace typu popisující chybové kódy vracené komunikačními objekty odvozenými od objektu tCoBase. Bližší popis lze najít v manuálu CoBase.

```
POverlapped = ^TOverlapped;
TOverlapped = record
  Reserved: array[0..5] of Byte;
end;
```

Struktura TOverlapped slouží k uchovaní stavu prováděné tzv. neblokující operace. Aplikace nesmí v žádném případě měnit obsah struktury TOverlapped po dobu provádění operace. Bližší popis lze najít v manuálu CoBase.

```
TString16 = String[16];
```

Deklarace typu zkráceného řetězce pro práci s knihovnami CoBase.

```
TString80 = String[80];
```

Deklarace typu zkráceného řetězce pro práci s knihovnami CoBase.

8. Objekty

8.1. tUDPPrt

8.1.1. Položky

Pozn.

1. Dále uvedené položky slouží pro vnitřní použití a neměly by být proto nastavovány přímo, pokud nebude uvedeno jinak.
2. Některé dále uvedené položky slouží při inicializaci komunikačních objektů, proto při jejich pozdějších změnách pomocí volání metod nebudou mít tyto položky aktuální hodnoty. Aktuální nastavení komunikačních objektů vždy získáte pomocí metody ChGetParam.

```
CH_Tick      : Boolean;
```

Položka se používá jako příznak o vykonávání činnost automatu kanálu.

```
CH_RTick     : Boolean;
```

Položka se používá jako příznak o vykonávání činnosti přijímacího automatu.

`CH_STick : Boolean;`
Položka se používá jako příznak o vykonávání činnosti vysílacího automatu.

`CH_DLE : Boolean;`
Položka slouží pro uložení příznaku zpracování znaku DLE.

`CH_SBuff : Pointer;`
Položka obsahuje ukazatel na vysílací buffer.

`CH_MSBuff : Word;`
Položka obsahuje délku vysílacího bufferu v bytech.

`CH_vMSBuff : Word;`
Položka obsahuje délku vysílaného rámce v bytech včetně redundance a řídících znaků.

`CH_LRMess : Word;`
Položka obsahuje velikost přijaté zprávy uložené v rámci v položce LEN.

`CH_vLRMess : Word;`
Položka obsahuje počet znaků v datové části rámce a jejich počet je určen stavem při dekódování rámce. Tato položka by se při úspěšném dekódování zprávy měla rovnat CH_LRMess.

`CH_RCrc : tCrc16;`
Položka obsahuje instanci objektu provádějící výpočet CRC u přijatého rámce. Počáteční residuum je 0 a na rozdíl od objektu tCrc16 umí provádět blokové operace.

`CH_SCrc : tCrc16;`
Položka obsahuje instanci objektu provádějící výpočet CRC u vysílaného rámce. Počáteční residuum je 0 a na rozdíl od objektu tCrc16 umí provádět blokové operace.

`CH_RAddr : TUDPAddress;`
Protože implicitně tato knihovna přijímá všechny pakety směřující na zadaný socket, musí se zapamatovat adresa přijatého paketu. Při odesílání odpovědi se pak použije tento socket.

`CH_RSocket : TUDPAddress;`
V této položce je uloženo nastavení socketu vzdálené stanici, které se použije při vytváření socketu. Implicitně je nastaven příjem všech dat UDP paketů na portu 5000 a tato položka je určena pro přímou editaci.

`CH_RecvLen : Word;`
Velikost dat přijatého UDP paketu. (Pozn. V těchto datech je uložen přijatý rámec).

`CH_Ptr : Word;`
Ukazatel na právě zpracovávaný znak přijatého rámce (dat UDP).

`CH_Socket : PCoDevice;`
Ukazatel na objekt spravující socket.

`CH_fCrStack : Boolean;`
V položce je uložen příznak, že zásobník zařízení tříd tCoBase byl vytvořen dříve a nemá být proto při uzavírání objektu uvolněn.

`CH_Status : TCoStatus;`
V položce je uložen status posledního volání funkce nižších vrstev objektu tCoUDP. Bližší popis těchto chybových kódů a jejich převodu na řetězec najdete v příslušných manuálech – CoBase, CoIPv4, CoUDP atd.

`CH_Timeout : LongInt;`

V této položce je zadán timeout pro operace s knihovnami CoBase. Tato položka je určena pro přímou editaci.

`CH_NicName : tString16;`

V této položce je zadán název objektu implementujícího ovladače pro síťovou kartu. Tato položka se použije při volání metody ChOpen pouze v případě, že ještě není vytvořen zásobník zařízení. Pokud je vytvořen, použije se současné nastavení. Implicitně je nastavena na *ETH01*. Tato položka je určena pro přímou editaci.

`CH_NicParam : tString80;`

V této položce jsou zadány parametry pro ovladač síťové karty. Tato položka se použije při volání metody ChOpen pouze v případě, že ještě není vytvořen zásobník zařízení. Pokud je vytvořen, použije se současné nastavení. Implicitně je nastavena na *‚IOBASE=$2300'* pro verzi MCP a *‚IOBASE=$300'* pro ostatní. Tato položka je určena pro přímou editaci.

`CH_IPParam : tString80;`

V této položce jsou zadány parametry pro IP vrstvu. Tato položka se použije při volání metody ChOpen pouze v případě, že ještě není vytvořen zásobník zařízení. Pokud je vytvořen, použije se současné nastavení.
Implicitně je nastavena
‚IPADDR="192.168.1.200" NETMASK="255.255.255.0"‘.
Tato položka je určena pro přímou editaci.

`CH_UDPParam : tString80;`

V této položce jsou zadány parametry pro UDP vrstvu. Tato položka se použije při volání metody ChOpen a otevírání socketu. Implicitně je nastavena na 'LPORT=5000'. Tato položka je určena pro přímou editaci.

`CH_Suffix : tString80;`

V této položce je zadán sufix zásobníku zařízení . Tato položka se používá při každé operaci se zásobníkem zařízení. Implicitně je nastavena na prázdný řetězec. Tato položka je určena pro přímou editaci.

```
CH_SendOvlp : tOverlapped;
CH_RecvOvlp : tOverlapped;
CH_CtrlOvlp : tOverlapped;
```

Struktury pro neblokující operace při příjmu, vysílaní a práci s komunikačním kanálem. Bližší popis lze najít v manuálu CoBase.

8.1.2. Metody

8.1.2.1. Init

`constructor Init;`

Konstruktor slouží k vytvoření a inicializaci instance komunikačního objektu. Ve svém těle volá nejdříve konstruktor předka tChnVirt a poté inicializuje své položky.

Po ukončení volání jsou položky nastaveny na následující hodnoty:

```
CH_Type    := ‘UDPPrt’;
CH_Name    := ‘UDPPrt’;
CH_NumName := ChNumName(CH_Type);
CH_SCtrl   := CHS_SendNoReady;
CH_RCtrl   := CHS_ReceiveNoReady;
CH_Tick    := False;
```

```
  CH_STick   := False;
  CH_RTick   := False;
  CH_DLE     := False;
  CH_SBuff   := nil;
  CH_MSBuff  := 0;
  CH_LRMess  := 0;
  CH_vLRMess := 0;
  CH_Node    := 0;
  CH_DNode   := 0;
  CH_RSNode  := 0;
  CH_RDNode  := 0;

  { parametry zarizeni CoBase }
  CH_Ptr      := 0;
  CH_Socket   := nil;
  CH_fCrStack := FALSE;
  CH_Status   := CST_SUCCESS;
  CH_Timeout  := 20*1000;     { ms = 20s }
  CH_NicName  :='ETH01';
 {$ifdef MCP}
  CH_NicParam :='IOBASE=$2300';
 {$else}
  CH_NicParam :='IOBASE=$300';
 {$endif}
  CH_IPParam  :='IPADDR="192.168.1.200" NETMASK="255.255.255.0"';
  CH_UDPParam :='LPORT=5000';
  CH_Suffix   :='';
  FillChar(CH_SendOvlp, sizeof(TOverlapped), 0);
  FillChar(CH_RecvOvlp, sizeof(TOverlapped), 0);
  FillChar(CH_CtrlOvlp, sizeof(TOverlapped), 0);

  with CH_RSocket do
  begin
    Size    := Sizeof(TUDPAddress);
    Address := IP_ADDR_ANY;
    Port    := 5000;
  end;

  CH_NumNameParents := CH_NumName;
  CH_Chn     := nil;
  CH_Ctrl    := CHS_Close;
  CH_State   := CHS_Close;
  CH_Result  := res_Ok; { stav kanalu – metoda ChResult }
  CH_RResult := res_Ok; { stav prijmu – metoda ChReceiveResult }
  CH_SResult := res_Ok; { stav vysilani – metoda ChSendResult }
  CH_RBuff   := nil;
  CH_MRBuff  := 0;
  CH_RMess   := nil;
  CH_MRMess  := 0;
```

8.1.2.2. ChInitParam

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

Konstruktor slouží ke zkrácenému vytvoření instance komunikačního objektu s nastavením parametrů komunikace. V těle se nejdříve volá konstruktor Init a poté metoda ChSetParam.

8.1.2.3. Done

```
destructor Done; virtual;
```

Destruktor Done slouží ke zrušení instance objektu. V těle metody se nejdříve provede uvolnění socketu a poté zrušení zásobníku zařízení, pokud ho daný objekt

vytvořil. Dále se v případě alokované paměti uvolní buffer pro příjem a vysílání zpráv. Nakonec se zavolá destruktor předka.

8.1.2.4. ChSetOneParam

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

Metoda slouží k dekódování a nastavení jednoho konkrétního parametru, který je zadán v parametru S. Tato metoda je volána při zpracování metody ChSetParam. V případě chyby se zpracování textového konfiguračního řetězce přeruší a funkce se ukončí. Po jejím ukončení by se vždy měla volat funkce ChResult, která vrací Res_Ok, pokud nedošlo k žádné chybě. V opačném případě vždy vrací Res_ErrParamStr. Chyby nižších vrstev lze získat v položce CH_Status viz příslušné manuály komunikačních knihoven– CoEth01, CoIPv4 a CoUDP.

V objektu tUDPPrt se touto metodou mohou nastavit následující parametry:

LSB=Size

Zadáním parametru LSB (Length of Send Buffer) dojde k alokování nového vysílacího bufferu CH_SBuff dané velikosti Size. Pokud předtím existoval nějaký buffer, je nejdříve uvolněn. Alokovaný buffer slouží pro transformaci vysílané zprávy, která je poté předána nižším vrstvám k odvysílání. Při transformaci je ke zprávě přidána výše popsaná hlavička a redundance. Velikost bufferu může nabývat hodnot 17 až 32750 byte.

LRB=Size

Zadáním parametru LRB (Length of Receive Buffer) dojde k alokování nového přijímacího bufferu CH_RBuff dané velikosti Size. Pokud předtím existoval nějaký jiný buffer, je nejdříve uvolněn. Alokovaný buffer slouží pro zpracování přijaté zprávy z nižších vrstev a po její transformaci (odstranění hlavičky, redundance a kontrola CRC) překopírování do buffer CH_RMess. Velikost bufferu může nabývat hodnot 8 až 65534 byte.

NOD=Node

Parametrem NOD ("Node") se určuje číslo (adresa) vlastní stanice CH_Node v komunikační síti. Node může nabývat hodnot 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. DNode může nabývat hodnot 0 až 255, přičemž hodnota 0 znamená, že zpráva je určena všem připojeným stanicím.

NAM=NIC „NIC parameters“

Pomocí tohoto parametru je možné nastavit parametry pro síťovou vrstvu. V případě, že objekt socket CH_Socket ještě neexistuje, jsou tyto parametry uloženy pouze do položky CH_NICParam a následně jsou použity při jeho vytváření. Pokud socket existuje, pokusí se tyto parametry nastavit. Nelze-li tyto parametry nastavit, vrací se chyba.

Pozn. Protože při nastavování těchto parametrů může dojít k chybě, je důležité nahlédnout do příslušných manuálů a zjistit za jakých podmínek lze tyto textové konfigurační parametry nastavovat.

NAM=IP „IP parameters“

Pomocí tohoto parametru je možné nastavit parametry pro IP vrstvu. V případě, že objekt socket CH_Socket ještě neexistuje, jsou tyto parametry uloženy pouze do položky CH_IPParam a následně jsou použity při jeho vytváření. Pokud socket existuje, pokusí se tyto parametry nastavit. Nelze-li tyto parametry nastavit, vrací se chyba.

Pozn. Protože při nastavování těchto parametrů může dojít k chybě, je důležité nahlédnout do příslušných manuálů a zjistit za jakých podmínek lze tyto textové konfigurační parametry nastavovat.

NAM=UDP „UDP parameters“

Pomocí tohoto parametru je možné nastavit parametry pro UDP vrstvu. V případě, že objekt socket CH_Socket ještě neexistuje, jsou tyto parametry uloženy pouze do položky CH_UDPParam a následně jsou použity při jeho vytváření. Pokud socket existuje, pokusí se tyto parametry nastavit. Nelze-li tyto parametry nastavit, vrací se chyba.

Pozn. Protože při nastavování těchto parametrů může dojít k chybě, je důležité nahlédnout do příslušných manuálů a zjistit za jakých podmínek lze tyto textové konfigurační parametry nastavovat.

Příklad:

Příklad ukazuje, jak je možné nastavit parametry komunikačního objektu tUDPPrt - LSB na hodnotu 1000, NODE na hodnotu 20, DNODE na hodnotu 30, v IP vrstvě parametr TTL na hodnotu 64 a v UDP vrstvě parametr LPORT na hodnotu 5000. Pro jejich nastavení se musí volat metoda ChSetParam, která při zpracování jednotlivých parametrů volá metodu ChSetOneParam příslušného objektu.

```
ChSetParam('NAM=UDPPRT LSB=1000 NOD=20 DNO=30 NAM=IP TTL=64 NAM=UDP
LPORT=5000');
```

8.1.2.5. ChGetParam

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

Při zadání parametru S různém od hodnot **NIC**, **IP** a **UDP** vrací tato metoda nastavené hodnoty parametrů komunikačního objektu. Jejich seznam je uveden v popisu metody ChSetOneParam.

Pokud parametr S nabývá některé z následujících hodnot, vrací se parametry příslušné vrstvy:

NIC – Metoda vrací parametry síťové vrstvy.

IP Metoda vrací parametry IP vrstvy.

UDP Metoda vrací parametry UDP vrstvy.

Podrobný popis vracených parametrů najdete v příslušných manuálech – CoBase, CoIPv4, CoUDP.

8.1.2.6. ChOpen

```
procedure ChOpen; virtual;
```

Před voláním této metody musí být vytvořeny buffery pro příjem a vysílání zpráv a kanál musí být uzavřen, např. ChState vrací CHS_Close. Pokud jsou tyto podmínky splněny, aktivuje se vytvoření komunikačního kanálu, které se dokončí

opakovaným voláním metody ChReady. Kanál je bezchybně vytvořen poté, co metoda ChReady vrátí CHS_Open. V tomto stavu sice komunikační kanál existuje, ale ještě není možné vysílat. Protože při vytváření komunikačního kanálu může dojít k chybě, musí se po metodě ChReady volat metoda ChResult, která může případně vracet tyto chybové kódy:

Res_ErrOpen, Res_ErrNoClose, Res_ErrStack a Res_ErrSocket.

Pozn. Po vytvoření komunikačního kanálu tzn. socketu použitím položek CH_NicName, CH_NicParam, CH_IPParam, CH_UDPParam a CH_Suffix se nastaví TIMEOUT pro jednotlivé operace knihoven CoBase dle nastavení položky CH_Timeout.

Doporučené volání této funkce:

```
{ otevreni komunikacniho kanalu }
ChOpen;
if ChResult <> res_Ok then Exit;
theCommTimer.SaveTime; { objekt tTimer – TIMEOUT operace }
while ChReady<>CHS_Open do
begin
  if ChResult <> res_Ok then Exit;
  if theCommTimer.TstTime(wReadyTime) then Exit;
  Wait(1); { pouze v pripade pouziti ReTOS }
end;
```

8.1.2.7. ChClose

```
procedure ChClose; virtual;
```

Před voláním této metody musí být vytvořen socket a kanál by neměl být uzavřen, např. ChState vrací hodnotu různou od CHS_Close. Pokud jsou tyto podmínky splněny, aktivuje se zrušení komunikačního kanálu, které se dokončí opakovaným voláním metody ChReady. Kanál je bezchybně uzavřen poté, co metoda ChReady vrátí CHS_Close. Protože při vytváření komunikačního kanálu může dojít k chybě, musí se po metodě ChReady volat metoda ChResult, která může případně vracet tyto chybové kódy:

Res_ErrChannelNoExist a Res_ErrNoConnect.

Doporučené volání této funkce:

```
ChClose;
if ChResult <> res_Ok then Exit;
theCommTimer.SaveTime; { objekt tTimer – TIMEOUT operace }
while ChReady<>CHS_Close do
begin
  if ChResult <> res_Ok then Exit;
  if theCommTimer.TstTime(wReadyTime) then exit;
  Wait(1); { pouze v pripade pouziti ReTOS }
end;
```

8.1.2.8. ChConnect

```
procedure ChConnect; virtual;
```

Před voláním této metody musí být vytvořen socket, buffery pro příjem a vysílání zpráv a kanál musí být otevřen, např. ChState vrací CHS_Open. Pokud jsou tyto podmínky splněny, aktivuje se připojení ke komunikačnímu kanálu, které se dokončí opakovaným voláním metody ChReady. Kanál je bezchybně připojen poté, co metoda ChReady vrátí CHS_Connect. V tomto stavu už lze data vysílat a přijímat, ChReceiveReady vrací CHS_ReceiveReady a ChSendReady vrací CHS_SendReady.

Protože při vytváření komunikačního kanálu může dojít k chybě, musí se po metodě ChReady volat metoda ChResult, která může případně vracet tyto chybové kódy:

Res_ErrChannelNoExist, Res_ErrNoConnect, Res_ErrNoOpen
a Res_ErrBind.

Doporučené volání této funkce:

```
{ pripojeni komunikacniho kanalu }
ChConnect;
if ChResult <> res_Ok then Exit;
theCommTimer.SaveTime; { objekt tTimer – TIMEOUT operace }
while ChReady<>CHS_Connect do
begin
  if ChResult <> res_Ok then Exit;
  if theCommTimer.TstTime(wReadyTime) then exit;
  Wait(1); { pouze v pripade pouziti ReTOS }
end;
```

8.1.2.9. ChDisConnect

```
procedure ChDisConnect; virtual;
```

Před voláním této metody musí být vytvořen socket a kanál musí být připojen, např. ChState vrací CHS_Connect. Pokud jsou tyto podmínky splněny, aktivuje se odpojení od komunikačního kanálu, které se dokončí opakovaným voláním metody ChReady. Kanál je bezchybně odpojen poté, co metoda ChReady vrátí CHS_DisConnect. V tomto stavu už nelze data vysílat a přijímat, ChReceiveReady vrací hodnotu CHS_ReceiveNoReady a ChSendReady vrací hodnotu CHS_SendNoReady. Protože při vytváření komunikačního kanálu může dojít k chybě, musí se po metodě ChReady volat metoda ChResult, která může případně vracet tyto chybové kódy:

Res_ErrChannelNoExist a Res_ErrUnbind.

Doporučené volání této funkce:

```
ChDisConnect;
if ChResult<>res_Ok then exit;
theCommTimer.SaveTime; { objekt tTimer – TIMEOUT operace }
while ChReady<>CHS_DisConnect do
begin
  if ChResult<>res_Ok then exit;
  if theCommTimer.TstTime(wReadyTime) then exit;
  Wait(1); { pouze v pripade pouziti ReTOS }
end;
```

8.1.2.10. PrepareMessage

```
function PrepareMessage(MessBuff:Pointer; MessLen:Word):Boolean;
```

Interní funkce volaná při vysílání dat. Jejím účelem je příprava rámce do vysílacího bufferu CH_SBuff z dat určených ukazatelem MessBuff o velikosti MessLen. Pokud velikost rámce překročí velikost bufferu CH_MSBuff, přestane se rámec generovat a funkce vrací FALSE. Pokud příprava rámce proběhne bez chyb, vrací se TRUE.

8.1.2.11. ChSend

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

Před voláním této metody musí být vytvořen socket a kanál musí být připojen, např. ChState vrací CHS_Connect. Pokud jsou tyto podmínky splněny, pokračuje se přípravou rámce a jeho odvysíláním, které se dokončí opakovaným voláním metody ChSendReady. Rámec je bezchybně odvysílán poté, co metoda ChSendReady vrátí CHS_SendReady. Protože při vysílání rámce může dojít k chybě, musí se po metodě ChSendReady volat metoda ChSendResult, která může případně vracet tyto chybové kódy:

Res_ErrChannelNoExist, Res_ErrNoConnect, Res_ErrBufferSize, Res_ErrMsgSize a Res_ErrSendBuffer.

Doporučené volání této funkce:

```
ChSend(pMess, MessLen);
if ChSendResult<>res_Ok then exit;
theSendTimer.SaveTime; { objekt tTimer – TIMEOUT operace }
while ChSendReady<>CHS_SendReady do
begin
  if ChSendResult<>res_Ok then exit;
  if theSendTimer.TstTime(wReadyTime) then exit;
  Wait(1); { pouze v pripade pouziti ReTOS }
end;
```

8.1.2.12. ChSendTick

```
procedure ChSendTick; virtual;
```

V metodě je implementován automat pro vysílání dat. Metoda je určena pro interní použití a je volána z metody ChSendReady.

8.1.2.13. ChSendReady

```
function ChSendReady:TChState; virtual;
```

Metoda způsobí provedení jednoho kroku vysílacího automatu voláním metody ChSendTick a poté vrací hodnotu v položce CH_SCtrl. Pokud kanál není připojen, např. ChState nevrací CHS_Connect, vrací se vždy CHS_SendNoReady.

Funkce se používá pro zjištění, jestliže už byla zpráva odvysílána, v tomto případě vrací CHS_SendReady.

8.1.2.14. ChSendFlush

```
procedure ChSendFlush; virtual;
```

Před voláním této metody musí být vytvořen socket a kanál musí být připojen, např. ChState vrací CHS_Connect. Pokud jsou tyto podmínky splněny, provede se vyprázdnění vysílacích bufferů a nastavení automatu vysílače pro vysílání nové zprávy. Protože při této metodě může dojít k chybě, musí se volat metoda ChSendResult, která může případně vracet tyto chybové kódy:

Res_ErrChannelNoExist a Res_ErrNoConnect.

8.1.2.15. ChReceive

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

Před voláním této metody musí být vytvořen socket a kanál musí být připojen, např. ChState vrací CHS_Connect. Pokud jsou tyto podmínky splněny, pokračuje se v příjmu rámce a jeho zpracování, které se dokončí opakovaným voláním metody ChReceiveReady. Rámec je bezchybně přijat poté, co metoda ChReceiveReady vrátí CHS_ReceiveReady. Data jsou uložena do bufferu, který se nastaví pomocí metody ChReceiveBuffer. Protože při příjmu rámce může dojít k chybě, musí se po metodě ChReceiveReady volat metoda ChReceiveResult, která může případně vracet tyto chybové kódy:

Res_ErrChannelNoExist, Res_ErrNoReceiveReady, Res_ErrNoConnect, Res_ErrRecvBuffer, Res_ErrSOH, Res_ErrETX, Res_ErrLen, Res_Err a Res_ErrCrc.

Doporučené volání této funkce:

```
ChReceive(Len);
if ChReceiveResult<>res_Ok then exit;
theRecTimer.SaveTime; { objekt tTimer – TIMEOUT operace }
while ChReceiveReady<>CHS_ReceiveReady do
begin
  if ChReceiveResult<>res_Ok then exit;
  if theRecTimer.TstTime(wReadyTime) then exit;
  Wait(1); { pouze v pripade pouziti ReTOS }
end;
```

8.1.2.16. ChReceiveTick

```
procedure ChReceiveTick; virtual;
```

V metodě je implementován automat pro příjem dat. Metoda je určena pro interní použití a je volána z metody ChReceiveReady.

8.1.2.17. ChReceiveReady

```
function ChReceiveReady:TChState; virtual;
```

Metoda způsobí provedení jednoho kroku přijímacího automatu voláním metody ChReceiveTick a poté vrací hodnotu v položce CH_RCtrl. Pokud kanál není připojen, např. ChState nevrací CHS_Connect, vrací se vždy CHS_ReceiveNoReady.

Funkce se používá pro zjištění, jestliže už byla nějaká zpráva přijata, v tomto případě vrací CHS_ReceiveReady.

8.1.2.18. ChReceiveFlush

```
procedure ChReceiveFlush; virtual;
```

Před voláním této metody musí být vytvořen socket a kanál musí být připojen, např. ChState vrací CHS_Connect. Pokud jsou tyto podmínky splněny, provede se vyprázdnění přijímacích bufferů a nastavení automatu přijímače pro příjem nové zprávy. Protože při této metodě může dojít k chybě, musí se volat metoda ChReceiveResult, která může případně vracet tyto chybové kódy:

Res_ErrChannelNoExist a Res_ErrNoConnect.

8.1.2.19. ChTick

```
procedure ChTick; virtual;
```

V metodě je implementován automat pro práci s komunikačním kanálem. Metoda je určena pro interní použití a je volána např. z metody ChReady.

8.1.2.20. ChState

```
function ChState:TChState; virtual;
```

Metoda provede krok automatu komunikačního kanálu voláním metody ChTick a vrátí naposledy dosažený stabilní stav komunikačního kanálu v položce CH_State. Pomocí metody ChState je možno provádět test na dosažení základních stabilních stavů automatu komunikačního kanálu.

8.1.2.21. ChReady

```
function ChReady:TChState; virtual;
```

Metoda provede krok automatu komunikačního kanálu voláním metody ChTick a vrátí jeho aktuální stav. Pomocí metody ChReady je možno také provádět test na dosažení základních stabilních stavů automatu.

8.1.2.22. ChGetNode procedura

```
procedure ChGetNode(var SNode, DNode: TNode); virtual;
```

Metoda uloží do proměnné SNode číslo (adresu) stanice, která zprávu odeslala, a do proměnné DNode číslo (adresu) stanice, pro kterou byla zpráva určena. Tato metoda se volá po přijetí zprávy metodou ChReceive.

8.1.2.23. SetTimeouts

```
procedure SetTimeouts;
```

Metoda nastaví timeout všech operací komunikačních knihoven odvozených od CoBase viz manuál CoBase struktura TCoDeviceTimeouts na hodnotu uvedenou v položce CH_Timeout.

8.2. tAddUdpPrt

Typ tAddUdpPrt je typem objektu, který slouží k definování prvku v seznamu správce komunikačních objektů. Objekt tAddUdpPrt je potomkem objektu tAddChnVirt.

8.2.1. Metody

8.2.1.1. ChInit funkce

```
function ChInit: pChnVirt;
```

Tato metoda slouží k vytvoření instance komunikačního objektu tUDPPrt a vrácení jeho ukazatele pro pozdější zpracování.

9. Příklad

Ideový příklad volání jednotlivých metod komunikačního objektu je uveden v knihovně ChnVirt.