ChnVirt

JEDNOTKA DEFINUJÍCÍ RODIČOVSKÝ PRVEK KOMUNIKAČNÍCH OBJEKTŮ

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 1.30

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: 18.04.2005
Datum posledního uložení dokumentu: 18.04.2005
(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	Wil		První vydání.
1.10	5.XX	Tum	19.05.2003	Úprava dokumentu dle ISO9000.
1.20	5.XX	Wil	13.01.2004	Úprava popisu některých algoritmů.
1.30	5.XX	Wil	18.04.2005	Přidána funkce ChnVirt_ResultToStr.

1.2. Účel dokumentu

Tento dokument slouží jako popis jednotky definující rodičovský prvek komunikačních objektů.

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 „ChnTypes“.

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 ChnVirt byla vytvořena s cílem vytvoření základního rodičovského objektu **tChnVirt** pro tvorbu konkrétních komunikačních objektů. V tomto objektu jsou definovány virtuální metody, jejich názvy a význam při konkrétních komunikačních akcích. Komunikační objekty dědičné od tohoto rodičovského komunikačního objektu vytvoří jen metody, které jsou specifické pro daný typ komunikace, nebo upraví těla stávajících metod dle potřeby. Tímto je vytvořeno jednotné rozhraní pro všechny komunikace a uživatel proto může psát aplikační program bez ohledu na konkrétní fyzickou přenosovou cestu.

Knihovna rovněž definuje rodičovský objekt **tAddChnVirt**, který zajistí, aby daný komunikační objekt (objekt tChnVirt) byl k aplikaci připojen a popřípadě zajistí vytvoření instance tohoto objektu. Po přilinkování této jednotky do aplikace (příkazem "uses ChnVirt"), se jméno objektu tChnVirt automaticky vloží do tzv. seznamu správců komunikačních objektů (viz 5. `ChnCollection`) pro případné použití.

Komunikační knihovny lze rozdělit na knihovny nižších vrstev, jejichž komunikační objekty zprostředkovávají fyzické spojení s daným přenosovým médiem, a na knihovny vyšších vrstev, jejichž komunikační objekty transformují předávaná data a předávají je objektům nižších vrstev. Určení, přes jaké komunikační objekty bude komunikace probíhat, je voleno až parametrem nastavovací metody (viz 6.1.2.4 `ChSetParam`) komunikačního objektu nejvyšší vrstvy po vytvoření instance tohoto objektu. Komunikační objekty je možno zřetězit a vytvářet tak množství typů přenosových cest. Proto je v paměti udržován seznam správců komunikačních objektů, s jehož pomocí je po zavolání nastavovací metody vytvořena daná komunikační vrstva. Uživatel takto tvořených komunikačních knihoven může až za chodu aplikace definovat, přes jaké komunikační vrstvy bude komunikace probíhat. Jsou-li dané komunikační knihovny k aplikaci přisestaveny (uvedeny v části `uses`), není třeba provádět překlad zdrojového tvaru. Vytvořený aplikační program dokáže komunikovat přes fyzické rozhraní RS232, RS485, telefonní modem nebo radiový modem bez změny programového vybavení.

Některé použité konstanty a typy jsou předdefinované v jednotce **ChnTypes**.

3.1. Terminologie

Pro snazší popis jednotlivých odborných výrazů byly zavedeny určité termíny, které se v dokumentaci komunikačních knihoven používají. Čtenář se tak bude moci v textu lépe orientovat. V této terminologii nejsou uvedeny termíny definované programovacím jazykem Pascal, jelikož se předpokládá, že je čtenář s těmito termíny obeznámen. Jsou zde popsány a vysvětleny termíny týkající se výhradně komunikace.

- knihovna
  Pojmem *knihovna* se míní jednotka (unit) jazyka Pascal.
- komunikační knihovna
  Komunikační knihovna je jednotka definující určité komunikační rozhraní tj. komunikační objekt. Její název má tvar ChnXXX, kde za XXX je dosazen jednoznačný název pro daný druh komunikace např.: COM, PRT apod.

- rodičovský komunikační objekt
  Pojmem *rodičovský komunikační objekt* nazýváme objekt tChnVirt, který je dědicem objektu tObject a je rodičovským objektem pro další komunikační objekty.

- komunikační objekt
  Pojmem *komunikační objekt* rozumíme objekt definující určité komunikační rozhraní, ať už fyzické či jen transformaci a formát dat (protokol). Tento objekt je dědicem od rodičovského komunikačního objektu tChnVirt nebo od jiného komunikačního objektu, který má rodičovský komunikační objekt jako jednoho ze svých předků.

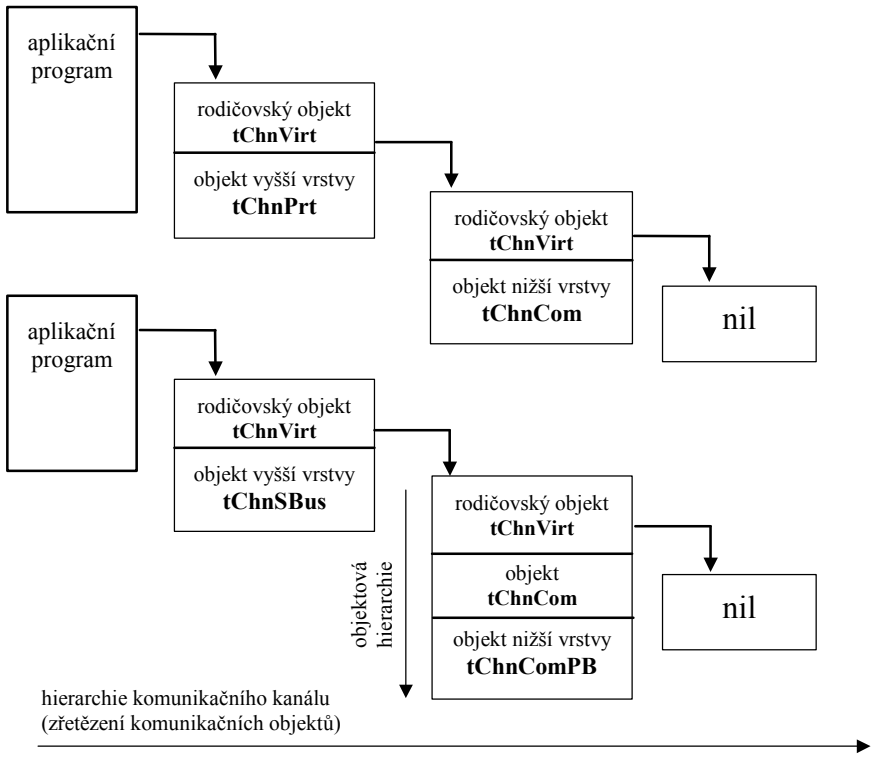
- komunikační kanál
  Komunikační kanál je lineární spojový seznam komunikačních objektů, kde komunikační objekt vyšší vrstvy definuje ukazatel na objekt nižší (podřízené) vrstvy. Např.: MyChn -> tChnPrt -> tChnCom -> nil. Dále se v dokumentaci používá výraz *zřetězený komunikační objekt*, kterým se míní komunikační objekt, který je v seznamu zapojen (zřetězen) za jiný komunikační objekt.

- komunikační vrstva
  Pojem *komunikační vrstva* se používá pro označení datového rozhraní mezi komunikačními objekty při implementaci protokolů. Většinou se v dokumentaci vyskytují pojmy *komunikační objekty vyšších vrstev* a naopak *komunikační objekty nižších vrstev*.

- seznam (objektů) správců komunikačních objektů
  Viz 5.`ChnCollection`.

3.2. Hierarchie komunikačních objektů

Pro názornost jsou schematicky uvedeny dva příklady komunikačních kanálů, na kterých je vidět rozdíl mezi objektovou hierarchií a hierarchií vzniklou zřetězením komunikačních objektů.



Z obrázku jsou patrné dvě linie. Horizontální, která představuje zřetězení jednotlivých objektů (vrstev) v komunikačním kanálu (lineární spojový seznam). Dále vertikální linie, která charakterizuje objektovou hierarchii a dědičnost (viz následující kapitola).

Aplikace volá pouze metody nejvyššího komunikačního objektu (vrstvy) z celého komunikačního kanálu. Tyto metody poté zajišťují nastavení/zjištění parametrů nižší vrstvy, případně jí předají příkaz a naopak z ní vrátí status. Při vlastním příjmu/vysílání dat metody vyšších vrstev provádějí konverze dat. Metody nejnižších vrstev poté provádějí vlastní vysílání a příjem přes komunikační rozhranní, tj. přistupují k hardware.

3.2.1. Objektová hierarchie

Pod tímto pojmem se rozumí vlastnost objektů v jazyce Pascal - dědičnost, kdy jeden objekt je schopen dědit položky a metody od svého rodičovského objektu. Ve výše uvedeném schématickém obrázku je tato objektová hierarchie naznačena v podobě svislého posazení dědičného objektu pod rodičovský objekt (např. objekt tChnComM je dědicem od tChnMod, který je dědicem od objektu tChnVirt).

3.2.2. Hierarchie vzniklá zřetězením komunikačních objektů

Pod tímto pojmem se rozumí lineární spojový seznam komunikačních objektů, kdy každý komunikační objekt obsahuje jako jednu ze svých položek ukazatel na instanci podřízeného komunikačního objektu. Ve výše uvedeném schématickém obrázku je toto lineární zřetězení naznačeno lomenými šipkami. Pomocí tohoto ukazatele předávají komunikační objekty vyšších vrstev řízení komunikačním objektům nižších vrstev.

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  = 'VIRT';
```

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

4.1. Konstanty stavů automatů

```
CHS_Close          = 0;
CHS_Open           = 1;
CHS_Connect        = 2;
CHS_DisConnect     = CHS_Open;
```

Konstanty definující základní stabilní stavy komunikačního kanálu.

Stav **CHS_Close** znamená, že kanál není otevřen. Tento stav je nastaven vždy po vytvoření instance komunikačního objektu.

Stav **CHS_Open** je dosažen po úspěšném volání metody **ChOpen**, která provede nastavení technického vybavení komunikačního kanálu.

Stav **CHS_Connect** je dosažen po úspěšném volání metody **ChConnect** a znamená, že je spojení navázáno a je možné po daném kanále komunikovat (přijímat a vysílat data pomocí metod **ChReceive** a **ChSend**).

Stav **CHS_DisConnect** je dosažen po volání metody **ChDisConnect** a znamená, že v komunikačním kanále je spojení přerušeno. Tento stav je shodný se stavem **CHS_Open**.

Tyto stavy jsou navraceny metodami **ChState** a **ChReady**. Kromě těchto základních stavů mohou konkrétní komunikační objekty definovat i přechodné stavy, které jsou využité při přechodu mezi jednotlivými stabilními stavy.

```
CHS_ReceiveReady         = 0;
CHS_ReceiveNoReady       = 1;
```

Konstanty definující základní stabilní stavy přijímače komunikačního kanálu.

Stav **CHS_ReceiveReady** oznamuje, že přijímač má k dispozici přijatá data.

Stav **CHS_ReceiveNoReady** naopak říká, že žádná přijatá data k dispozici nejsou.

Po vytvoření instance objektu je přijímač ve stavu **CHS_ReceiveNoReady**.

Tyto stavy jsou navraceny metodou **ChReceiveReady**. Kromě těchto základních stavů mohou konkrétní komunikační objekty definovat i přechodné stavy, které jsou využité při přechodu mezi jednotlivými stabilními stavy.

```
CHS_SendReady           = 0;
CHS_SendNoReady         = 1;
```

Konstanty definující základní stabilní stavy vysílače komunikačního kanálu.

Stav **CHS_SendReady** oznamuje, že vysílač odvysílal všechny požadované zprávy a je připraven vysílat.

Stav **CHS_SendNoReady** naopak říká, že vysílač dosud neodvysílal požadovaná data a není připraven vysílat další zprávy.

Po vytvoření instance objektu je nastaven stav **CHS_SendReady**. Tyto stavy jsou navraceny metodou **ChSendReady**. Kromě těchto základních stavů mohou konkrétní komunikační objekty definovat i přechodné stavy, které jsou využité při přechodu mezi jednotlivými stabilními stavy.

4.2. Konstanty výsledků operací

```
res_Ok                  = $0000;

res_ErrNoReceiveReady   = $00c0;
res_ErrNoRecBuff        = $00c1;
res_ErrSmallRecBuff     = $00c2;

res_ErrNoSendReady      = $00d0;
res_ErrNoSendBuff       = $00d1;
res_ErrSmallSendBuff    = $00d2;

res_ErrNoClose          = $00e0;
res_ErrNoOpen           = $00e1;
res_ErrNoConnect        = $00e2;
res_ErrOpen             = $00e3;
res_ErrConnect          = $00e4;
res_ErrDisConnect       = $00e5;
res_ErrClose            = $00e6;

res_ErrIllegalUnit      = $00fa;
res_ErrChannelNoExist   = $00fb;
res_ErrParamStr         = $00fc;
res_Err                 = $00ff;
```

Konstanty definující základní výsledky po provedení operací s přenosovým kanálem.

Výsledek **res_Ok** znamená, že operace proběhla bez chyb.

Výsledek **res_ErrNoReceiveReady** znamená, že při volání metod **ChReceive** nebo **ChReceiveChar** přijímač nemá k dispozici žádná přijatá data. Proto se nejdříve aplikace musí prostřednictvím metody **ChReceiveReady** dotázat na stav přijímače a teprve poté (v případě přijetí dat) volat metody **ChReceive** a **ChReceiveChar**.

Výsledek **res_ErrNoRecBuff** znamená, že není definován přijímací buffer. Aplikace zapomněla tento buffer nastavit příslušným parametrem nastavovací metody ChSetParam (zpravidla se jedná o parametr s názvem LRB).

Výsledek **res_ErrSmallRecBuff** znamená, že při příjmu nastal pokus o příjem větší zprávy, než která se vejde do přijímacího bufferu. Aplikace nastavila příslušným parametrem nastavovací metody ChSetParam (zpravidla se jedná o parametr s názvem LRB) příliš malý buffer.

Výsledek **res_ErrNoSendReady** znamená, že při volání metody **ChSend** vysílač není připraven odvysílat zadaná data (dosud se neodvysílala předchozí zadaná data).

Výsledek **res_ErrNoSendBuff** znamená, že není definován vysílací buffer pro objekt nižší komunikační vrstvy. Tj. aplikace zapomněla tento buffer nastavit příslušným parametrem nastavovací metody ChSetParam (zpravidla se jedná o parametr s názvem LSB nebo LTB). Pozn: tento výsledek se používá pouze u komunikačních objektů vyšších vrstev.
Výsledek **res_ErrSmallSendBuff** znamená, že při vysílání nastal pokus o vyslání větší zprávy, než která se vejde do vysílacího bufferu pro objekt nižší komunikační vrstvy. Tj. aplikace nastavila tento buffer příslušným parametrem nastavovací metody ChSetParam (zpravidla se jedná o parametr s názvem LSB nebo LTB) na příliš malou hodnotu. Pozn: tento výsledek se používá pouze u komunikačních objektů vyšších vrstev.
Výsledek **res_ErrNoClose** znamená, že kanál není ve stavu **CHS_Close**.
Výsledek **res_ErrNoOpen** znamená, že kanál není ve stavu **CHS_Open**.
Výsledek **res_ErrNoConnect** znamená, že kanál není ve stavu **CHS_Connect**.
Výsledky **res_ErrOpen**, **res_ErrConnect**, **res_ErrDisConnect** a **res_ErrClose** znamenají, že metody **ChOpen**, **ChConnect**, **ChDisConnect** nebo **ChClose** neproběhly úspěšně.
Výsledek **res_ErrIllegalUnit** znamená, že následující komunikační vrstva, která byla zřetězena v komunikačním kanálu, je nepřípustná. Například nemůžete zřetězit komunikační vrstvu **ChnPrt** za vrstvu **ChnCom** a podobně.
Výsledek **res_ErrChannelNoExist** znamená, že metoda se pokouší odkazovat na následující zřetězený komunikační objekt v komunikačním kanálu, jehož instance není vytvořena.
Výsledek **res_ErrParamStr** znamená, že volání metody **ChSetParam** skončilo chybou.
Výsledek **res_Err** znamená, že volání metody skončilo blíže neurčenou chybou.

Kromě těchto základních výsledků mohou dané komunikační knihovny definovat vlastní výsledky, které blíže určí detekovanou chybu. Kódy těchto nových výsledků nesmí být shodné s kódy již dříve definovaných výsledků. Výše uvedené výsledky jsou navraceny v nižším byte metodami **ChResult, ChReceiveResult** a **ChSendResult**. Ve vyšším byte je navraceno číselné jméno komunikačního objektu, který chybu zaznamenal, pokud výsledek je různý od **res_Ok**.
Pro přehlednější zjištění popisu chyby lze použít funkce ChnVirt_ResultToStr (viz „7.1 ChnVirt_ResultToStr funkce“).

4.3. Typy

```
tChName     = tString10;
```
Typ **tChName** definuje typ jména kanálu.

```
tChNumName  = Word;
```
Typ **tChNumName** definuje typ číselného jména kanálu.

```
tChResult   = Word;
```
Typ **tChResult** definuje typ výsledku operace.

```
tChResultStr = string[32];
```
Typ **tChResultStr** definuje typ textové representace výsledku operace.

`tChState = Word;`

Typ **tChState** definuje typ stavu kanálu.

`tNode = Word;`

Typ **tNode** definuje typ čísla (adresy) stanice v komunikační síti.

5. Proměnné

`ChnCollection : PChnCollection = nil;`

Ukazatel na seznam objektů správců komunikačních objektů (dále jen "seznam správců"). Do tohoto seznamu jsou vkládány objekty správců komunikačních objektů. Objekt správce komunikačního objektu obsahuje jméno spravovaného komunikačního objektu a je schopen vytvořit novou instanci tohoto komunikačního objektu.

Tuto proměnnou zpravidla používají metody komunikačních objektů pro vyhledávání dalších komunikačních objektů v komunikačním kanálu.

6. Objekty

6.1. tChnVirt

6.1.1. Položky

`CH_Type : TChName;`

Položka **CH_Type** uchovává jméno typu daného komunikačního objektu. Po vytvoření instance daného objektu je tato položka shodná se jménem definovaným pro danou komunikační knihovnu a dále se v aplikaci nesmí měnit.

`CH_Name : TChName;`

Položka **CH_Name** uchovává jméno instance komunikačního objektu. Po vytvoření instance daného objektu je toto jméno shodné se jménem typu daného komunikačního objektu (CH_Type), ale dále jej může uživatel v aplikaci libovolně měnit.

`CH_NumName : TChNumName;`

Položka **CH_NumName** uchovává číselné jméno typu komunikačního objektu. Toto číselné jméno je definováno po vytvoření instance daného objektu a je poplatné pro danou aplikaci. Konverzi mezi textovým jménem **CH_Type** a číselným jménem **CH_NumName** zajišťují metody **ChName** a **ChNumName**. Tato položka byla zavedena pro rychlejší dekódování přístupu k danému komunikačnímu objektu. Některé metody potřebují jako parametr zadat, ke kterému komunikačnímu objektu, který byl zřetězen v komunikačním kanálu, mají přistupovat. Kdyby se jako parametr uvádělo textové jméno komunikačního objektu, trvalo by dekódování tohoto jména podstatně déle, než dekódování jeho číselné podoby.

`CH_NumNameParents: TChNumName;`

Položka **CH_NumNameParents** uchovává číselné jméno typu přímého rodičovského komunikačního objektu dané komunikační knihovny. Po vytvoření instance je nastaveno na číselné jméno objektu **tChnVirt**

(ChNumName (ChnVirt.cName)), který je prvním rodičovským komunikačním objektem pro všechny ostatní komunikační objekty. Komunikační knihovny definující komunikační objekty, jež nejsou přímými dědici rodičovského komunikačního objektu **tChnVirt**, musí tuto proměnou předefinovat na základě číselného jména typu přímého rodičovského komunikačního objektu (ChNumName (ChnXXX.cName)).

`CH_Chn : PChnVirt;`

Položka **CH_Chn** uchovává ukazatel na instanci následujícího zřetězeného komunikačního objektu v komunikačním kanálu. Při vytvoření instance objektu je implicitně nastavena na NIL.

`CH_Ctrl : TChState;`

Položka **CH_Ctrl** uchovává aktuální stav automatu kanálu, tj. kanál otevřený, uzavřený, spojení navázáno, spojení ukončeno, přechází, atd. (viz výše definované konstanty CHS_Xxxx). Její obsah je navracen metodou **ChReady**.

`CH_RCtrl : TChState;`

Položka **CH_RCtrl** uchovává aktuální stav automatu přijímače, tj. přijímač má k dispozici přijatá data (CHS_ReceiveReady), přijímač nemá k dispozici přijatá data (CHS_ReceiveNoReady), přechází, atd. Její obsah je navracen metodou **ChReceiveReady**.

`CH_SCtrl : TChState;`

Položka **CH_SCtrl** uchovává aktuální stav automatu vysílače, tj. vysílač je připraven vysílat (CHS_SendReady), dosud neodvysílal požadovaná data (CHS_SendNoReady), přechází, atd. Její obsah je navracen metodou **ChSendReady**.

`CH_State : TCHState;`

Položka **CH_State** udržuje poslední dosažený stabilní stav automatu kanálu, tj. kanál otevřený (CHS_Open), uzavřený (CHS_Close), spojení navázáno (CHS_Connect), spojení ukončeno (CHS_DisConnect), atd. Její obsah je navracen metodou **ChState**.

`CH_Result : TChResult;`

Do položky **CH_Result** je uložen výsledek vzniklý voláním metod komunikačního objektu souvisejících s komunikačním kanálem. Je uchováván až do následného volání metody **ChResult**, která navrací hodnotu této položky a poté ji nastaví na **res_Ok**.

`CH_RResult : TChResult;`

Do položky **CH_RResult** je uložen výsledek vzniklý voláním metod komunikačního objektu souvisejících s operacemi příjmu dat. Je uchováván až do následného volání metody **ChReceiveResult**, která navrací hodnotu této položky a poté ji nastaví na **res_Ok**.

`CH_SResult : TChResult;`

Do položky **CH_SResult** je uložen výsledek vzniklý voláním metod komunikačního objektu souvisejících s operacemi vysílání dat. Je uchováván až do následného volání metody **ChSendResult**, která navrací hodnotu této položky a poté ji nastaví na **res_Ok**.

`CH_Node : TNode;`

CH_Node je číslo (adresa) stanice v dané komunikační síti. U komunikací, které nejsou typu lokální síť, není využíváno. Tato položka se nastavuje pomocí metody **ChSetParam**.

`CH_DNode : TNode;`

CH_DNode je číslo (adresa) stanice adresáta zprávy v dané komunikační síti, kterému bude určena příští zpráva. U komunikací, které nejsou typu lokální síť, není využíváno. Tato položka se nastavuje metodou **ChDestNode**.

`CH_RSNode : TNode;`

Do položky **CH_RSNode** je ukládáno při příjmu zprávy číslo (adresa) stanice v dané komunikační síti, která tuto zprávu poslala. U komunikací, které nejsou typu lokální síť, není využívána.

`CH_RDNode : TNode;`

Do položky **CH_RDNode** je ukládáno při příjmu zprávy číslo (adresa) stanice v dané komunikační síti, které je zpráva určena. U komunikací, které nejsou typu lokální síť, není využívána.

Položky **CH_RSNode** a **CH_RDNode** lze přečíst po přijetí zprávy pomocí metody **ChGetNode**.

`CH_RBuff : Pointer;`

CH_RBuff je ukazatel na přijímací vyrovnávací buffer, do kterého jsou při příjmu postupně ukládána příchozí data.

`CH_MRBuff : Word;`

Položka **CH_MRBuff** určuje maximální velikost přijímacího vyrovnávacího bufferu.

Položky **CH_RBuff** a **CH_MRBuff** se nastavují pomocí metody **ChSetParam**.

`CH_RMess : Pointer;`

Položka **CH_RMess** je ukazatel na proměnnou v aplikačním programu, do které má být přijatá zpráva metodou **ChReceive** uložena. Položka je nastavována metodou **ChReceiveBuffer**.

`CH_MRMess : Word;`

CH_MRMess je maximální velikost proměnné v aplikačním programu, do které má být přijatá zpráva uložena. Položka je nastavována metodou **ChReceiveBuffer**.

6.1.2. Metody

6.1.2.1. Init konstruktor

`constructor Init;`

Konstruktor **Init** slouží k vytvoření a inicializaci instance komunikačního objektu. Položky objektu jsou nastaveny na implicitní hodnoty a aplikace může volat metody komunikačního objektu. V těle konstruktoru jsou položky objektu inicializovány následovně :

```
{ 1} inherited Init;     { je volán Init rodičovského objektu }
{ 2} CH_Type             := cName;
{ 3} CH_Name             := CH_Type;
{ 4} CH_NumName          := ChNumName(CH_Type);
{ 5} CH_NumNameParents   := CH_NumName;
{ 6} CH_Chn              := nil;
{ 7} CH_Ctrl             := CHS_Close;
{ 8} CH_RCtrl            := CHS_ReceiveNoReady;
{ 9} CH_SCtrl            := CHS_SendReady;
{10} CH_State            := CHS_Close;
{11} CH_Result           := res_Ok;
{12} CH_RResult          := res_Ok;
```

```
{13} CH_SResult         := res_Ok;
{14} CH_Node            := 0;
{15} CH_DNode           := 0;
{16} CH_RSNode          := 0;
{17} CH_RDNode          := 0;
{18} CH_RBuff           := nil;
{19} CH_MRBuff          := 0;
{20} CH_RMess           := nil;
{21} CH_MRMess          := 0;
```

Každý další komunikační objekt musí předefinovat konstruktor **Init** tak, že v těle svého konstruktoru nejprve zavolá zděděný konstruktor (inherited Init - řádek 1) od přímého rodičovského komunikačního objektu pro inicializaci položek definovaných v tomto rodičovském objektu, a poté provede inicializaci vlastních položek. Musí se také nově nainicializovat položky **CH_Type**, **CH_Name** a **CH_NumName** pro nastavení aktuálního jména své komunikační vrstvy. To se provede stejným způsobem, tak jak je ukázáno výše (řádky 2 až 4). Pokud komunikační objekt není přímým dědicem od rodičovského komunikačního objektu tChnVirt musí předefinovat i položku **CH_NumNameParents** a to následovně: CH_NumNameParents := ChNumName (ChnXxx.cName);, kde ChnXxx je název komunikační knihovny, která definuje přímý rodičovský komunikační objekt. Souhrnem těchto podmínek pro definování konstruktorů dalších komunikačních objektů dostaneme, že konstruktor nově definovaného objektu musí mít řádky 1 až 4 syntakticky shodné s výše vedeným výpisem těla konstruktoru tChnVirt.Init. Následující příklad ukazuje způsob definování konstruktorů dalších komunikačních objektů:

```
constructor tChnXxx.Init;
...
begin
  inherited Init;
  CH_Type    := cName;
  CH_Name    := CH_Type;
  CH_NumName := ChNumName(CH_Type);
  ...
  { inicializace vlastních položek }
  ...
end;
```

6.1.2.2. ChInitParam konstruktor

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

Konstruktor ChInitParam je sloučením konstruktoru **Init** a metody **ChSetParam**. Při vytváření dědičných komunikačních objektů je nutno vždy konstruktor **ChInitParam** předefinovat tak, že tělo nového konstruktoru bude syntakticky shodné s tělem konstruktoru tohoto komunikačního objektu.

Vzhledem k vlastnostem konstruktorů v jazyce Pascal je toto předefinování nezbytné z důvodu vytvoření správné vazby do tabulky virtuálních metod (VMT).

6.1.2.3. Done destruktor

```
destructor  Done;
```

Destruktor **Done** slouží ke zrušení instance komunikačního objektu. Pokud kanál není ve stavu **CHS_Close**, provede se uzavření komunikačního kanálu voláním metody **ChClose**. Poté se provede zrušení instance podřízeného zřetězeného

komunikačního objektu, a pokud je alokován přijímací buffer či jiné datové struktury, budou odstraněny z paměti.

Další komunikační objekty předefinovávají destruktor **Done** tak, že v těle svého destruktoru nejprve provedou akce pro zrušení svých datových struktur a uzavřou svoji vrstvu kanálu a po provedení těchto akcí zavolají zděděný destruktor (inherited Done) od přímého rodičovského komunikačního objektu. Volání zděděného destruktoru je nezbytné pro zrušení (uvolnění paměti, uzavření kanálu) podřízené komunikační vrstvy. Komunikační objekty, které neprovádějí žádné uzavření své komunikační vrstvy ani zrušení svých datových struktur (volaly by pouze "inherited Done"), nemusí destruktor **Done** vůbec definovat. Následující příklad ukazuje způsob definování destruktorů dalších komunikačních objektů:

```
destructor tChnXxx.Done;
...
begin
  ...
  { zrušení vlastních datových struktur a
    uzavření své vrstvy komunikačního kanálu }
  ...
  inherited Done;
end;
```

6.1.2.4. ChSetParam procedura

```
procedure ChSetParam(const S: TParamStr);
```

Metoda **ChSetParam** slouží k novému nastavení parametrů komunikačního kanálu. Není třeba definovat pokaždé hodnoty všech parametrů, ale stačí uvést jen ty parametry, kterých se změna bude týkat. Jsou-li komunikační objekty zřetězeny, musí být parametry pro následující zřetězené objekty nejprve jednoznačně určeny jménem zřetězeného komunikačního objektu. Poté jsou jim parametry předány pomocí private metody **ChSetNextParam**. Pro dekódování jednotlivých parametrů je volána metoda **ChSetOneParam** (viz následující kapitola).

Další komunikační objekty nemusí metodu **ChSetParam** vůbec předefinovávat, ani ji volat jako zděděnou (inherited).

Příklad:

Příklad ukazuje, jak je možné nastavit v komunikačním objektu se jménem PRT parametry xxx (resp. yyy) na hodnotu aaa (resp. bbb) a v komunikačním objektu se jménem COM, který byl zřetězen v komunikačním kanálu, parametr BD na hodnotu 9600.

```
ChSetParam('NAM=PRT XXX=aaa YYY=bbb | NAM=COM BD=9600');
```

Znak "|" slouží jako pomocný oddělovač mezi jednotlivými komunikačními objekty. Je nepovinný, jelikož určení, kterému komunikačnímu objektu jsou parametry určeny, je řečeno parametrem NAM. To znamená, že parametr NAM musí být vždy pro objekt dané komunikační vrstvy uveden jako první. Pokud je použit znak "|" za kterým parametr NAM nenásleduje, jsou parametry určeny pro následující zřetězený komunikační objekt (Toto je zachováno jen kvůli zpětné kompatibilitě se staršími projekty. Pro nové projekty není používání znaku "|" bez následujícího NAM čisté).

6.1.2.5. 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 je volána prostřednictvím metody **ChSetParam**. Jako svoji funkční hodnotu vrací výsledek úspěšnosti operace.

Aplikace by tuto metodu nikdy neměla volat přímo, ale pouze prostřednictvím metody **ChSetParam**. Ostatní komunikační knihovny by ji měly předefinovat vždy pro dekódování svých parametrů. Při vytváření dědičných komunikačních objektů není třeba znova dekódovat ty parametry, které umí dekódovat metoda **ChSetOneParam** rodičovského komunikačního objektu. Stačí dekódovat nové parametry a při nalezení neznámého parametru zavolat metodu **ChSetOneParam** rodičovského komunikačního objektu (inherited) pro případ, že by ona neznámý parametr dekódovat uměla. Níže uvedené parametry, které dekóduje metoda **ChSetOneParam** komunikačního objektu tChnVirt, jsou tedy společné pro všechny další komunikační objekty odvozené od rodičovského objektu tChnVirt a každý z těchto objektů je může používat. Metoda **ChSetOneParam** komunikačního objektu tChnVirt dekóduje tyto parametry:

TYP=Type

Parametrem **TYP** ("Type Name") se určuje, že následující parametry jsou určeny komunikačnímu objektu se jménem typu Type (objektu, který má položku CH_Type = Type).

NAM=Name

Parametrem **NAM** ("Name") se určuje, že následující parametry jsou určeny komunikačnímu objektu se jménem nebo jménem typu Name (objektu, který má položku CH_Type = Name nebo položku CH_Name = Name).

DNA=NewName

Parametrem **DNA** ("Define Name") se nastaví hodnota jména komunikačního objektu CH_Name na hodnotu NewName. Bez předcházejícího použití parametru CNA se smí tento úkon provést pro daný komunikační objekt pouze jednou, to znamená, že se parametr DNA nesmí použít dvakrát za sebou bez použití parametru CNA mezi.

CNA

Parametrem **CNA** ("Clear Name") se obnoví jméno komunikačního objektu CH_Name na původní hodnotu CH_Type a lze jej opětovně předefinovat pomocí parametru DNA.

DON

Parametrem **DON** ("Done") se zruší instance podřízeného komunikačního objektu, který byl zřetězen v komunikačním kanálu.

NIL

Parametrem **NIL** ("Nil") se zruší odkaz na instanci podřízeného komunikačního objektu, který byl zřetězen v komunikačním kanálu, ale instance tohoto objektu se nezruší a zůstane dostupná pomocí *seznamu správců*.

Pozn: Názvy jednotlivých parametrů mohou být zadány jak malými tak velkými písmeny, tj. metoda ChSetOneParam nerozlišuje velikost písmen.

6.1.2.6. ChSetNextParam procedura

```
procedure ChSetNextParam(const S: TParamStr);
```

Metoda **ChSetNextParam** slouží pro předání parametrů následujícímu zřetězenému komunikačnímu objektu. Je volána v metodě **ChSetParam**. Ve svém těle volá metodu **ChSetParam** následujícího zřetězeného objektu.

Aplikace by tuto metodu nikdy neměla volat přímo, ale pouze prostřednictvím metody **ChSetParam**. Další komunikační objekty ji nemusí předefinovávat ani volat jako zděděnou (inherited).

6.1.2.7. ChGetParam funkce

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

Metoda **ChGetParam** navrací nastavené hodnoty parametrů daného komunikačního objektu. Parametr `S` určuje, z kterého objektu a které parametry (popřípadě kterou sadu parametrů) bude metoda navracet. Pokud parametr `S` je prázdný řetězec (S = ''), metoda se pokusí vrátit nastavení všech parametrů daného komunikačního objektu, včetně parametrů dalších zřetězených komunikačních objektů. Každý další komunikační objekt by měl tuto metodu předefinovat pro vrácení svých parametrů a poté vrátit i parametry dalšího zřetězeného komunikačního objektu.

6.1.2.8. ChSetBinParam procedura

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

Metoda **ChSetBinParam** umožňuje nastavení libovolných parametrů komunikace v binárním tvaru, které jsou poplatné pouze danému komunikačnímu objektu s číselným jménem `NumName`. Parametr `Code` určuje prováděnou akci a parametr `Param` určuje potřebná data (může to být také ukazatel na datový buffer). Tato metoda může na základě parametru `Code` nastavovat hodnoty určitých položek v daném komunikačním objektu nebo může provádět určité akce (jako je např. čtení modemových signálů apod.). Komunikační objekty, které nevyžadují tímto způsobem provádět určité akce či nastavovat parametry, nemusí tuto metodu vůbec definovat.

6.1.2.9. ChGetBinParam funkce

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

Metoda **ChGetBinParam** umožňuje vracet nastavení libovolných parametrů komunikace v binárním tvaru, které jsou poplatné pouze danému komunikačnímu objektu s číselným jménem `NumName`. Parametr `Code` určuje, prováděnou akci. Metoda vrací jako svou funkční hodnotu příslušná data (může vracet i ukazatel na datový buffer). Tato metoda může na základě parametru `Code` nejenom vracet nastavení určitých položek komunikačního objektu, ale může vracet i status či stav určité akce, která byla vyvolána metodou **ChSetBinParam**, popřípadě provedení dalšího kroku akce apod. Knihovny, které nevyžadují tímto způsobem provádět určité akce či nastavovat parametry, nemusí tuto metodu vůbec definovat.

6.1.2.10. ChNumName funkce

```
function  ChNumName(Name: tChName): tChNumName;
```

Metoda **ChNumName** převádí zadané textové jméno komunikačního objektu na jeho číselnou podobu. Výsledek pak vrací jako svoji funkční hodnotu. Jako parametr se zadává jméno typu daného komunikačního objektu. Tuto metodu ostatní komunikační objekty dále nepředefinovávají.

6.1.2.11. ChName funkce

```
function  ChName(NumName: tChNumName): tChName;
```

Metoda **ChName** převádí zadané číselné jméno komunikačního objektu na jeho textovou podobu. Výsledek pak vrací jako svoji funkční hodnotu. Jako parametr se zadává číselné jméno typu daného komunikačního objektu. Tuto metodu ostatní komunikační objekty dále nepředefinovávají. Je to inverzní metoda k metodě **ChNumName**.

6.1.2.12. ChAllNumName funkce

```
function  ChAllNumName: String;
```

Metoda **ChAllNumName** vrátí řetězec textových a číselných jmen typů všech komunikačních objektů, které jsou pro danou aplikaci dostupné pomocí *seznamu správců*. Tuto metodu ostatní komunikační objekty dále nepředefinovávají.

6.1.2.13. ChAllName funkce

```
function  ChAllName: String;
```

Metoda **ChAllName** vrátí řetězec textových jmen typů všech komunikačních objektů, které jsou pro danou aplikaci dostupné pomocí *seznamu správců*. Tuto metodu ostatní komunikační objekty dále nepředefinovávají.

6.1.2.14. ChOpen procedura

```
procedure ChOpen;
```

Po vytvoření instance komunikačního objektu je kanál ve stavu **CHS_Close**. Není nastaveno technické vybavení. Metodou **ChSetParam** můžeme nastavit položky komunikačního objektu a parametry komunikace a metodou **ChOpen** inicializovat technické vybavení. Po volání metody **ChOpen** komunikační kanál přejde po čase do stavu **CHS_Open** (pokud bylo volání metody ChOpen úspěšné).

Komunikační objekty nižších vrstev by měly tuto metodu zpravidla celou předefinovat na základě použitého technického vybavení, zatímco komunikační objekty vyšších vrstev tuto metodu zpravidla nepředefinovávají.

Příklad správného volání metody **ChOpen**:

```
ChOpen;  {zavolání metody ChOpen}
repeat
  if ChResult<>res_Ok then ...; {akce pro ošetření chyby}
until ChReady = CHS_Open;  {čeká se do dosažení stavu CHS_Open}
if ChResult<>res_Ok then ...;   {akce pro ošetření chyby}
```

Pokud aplikace používá operačního systému reálného času (o.s.Retos), měl by být v těle cyklu použit příkaz `Wait(1);` pro případné předání řízení i ostatním procesům.

Stejným způsobem by měly být volány i metody **ChClose**, **ChConnect** a **ChDisConnect**.

6.1.2.15. ChClose procedura

```
procedure ChClose;
```

Metoda **ChClose** uzavře komunikační kanál a způsobí přechod do stavu **CHS_Close**. Metoda by měla uvést technického vybavení kanálu do neaktivního stavu.

Komunikační objekty nižších vrstev by měly tuto metodu zpravidla celou předefinovat na základě použitého technického vybavení, zatímco komunikační objekty vyšších vrstev tuto metodu zpravidla nepředefinovávají.

Po volání této metody lze opětovně volat metodu **ChOpen**. Příklad správného volání této metody je podobný příkladu volání metody **ChOpen**.

6.1.2.16. ChConnect procedura

```
procedure ChConnect;
```

Metoda **ChConnect** způsobí navázání komunikace mezi účastníky (např. navázání telefonního spojení). Před voláním metody musí být kanál ve stavu **CHS_Open**. Po úspěšném volání metody přejde kanál po čase do stavu **CHS_Connect**, to znamená, že je možno po daném kanále komunikovat (posílat a přijímat data pomocí metod **ChSend** a **ChReceive**).

Komunikační objekty nižších vrstev by měly tuto metodu zpravidla celou předefinovat na základě použitého technického vybavení, zatímco komunikační objekty vyšších vrstev tuto metodu zpravidla nepředefinovávají.

Příklad správného volání této metody je podobný příkladu volání metody **ChOpen**.

6.1.2.17. ChDisConnect procedura

```
procedure ChDisConnect;
```

Metoda **ChDisConnect** ukončí navázané komunikační spojení a uvede kanál do stavu **CHS_DisConnect**. Je přerušen příjem a vysílání zpráv.

Komunikační objekty nižších vrstev by měly tuto metodu zpravidla celou předefinovat na základě použitého technického vybavení, zatímco komunikační objekty vyšších vrstev tuto metodu zpravidla nepředefinovávají.

Po volání této metody lze opětovně volat metodu **ChConnect**. Příklad správného volání této metody je podobný příkladu volání metody **ChOpen**.

6.1.2.18. ChState funkce

```
function  ChState: TChState;
```

Metoda **ChState** provede krok automatu komunikačního kanálu na základě volání metody **ChTick** a navrátí naposled dosažený stabilní stav komunikačního kanálu (viz **Chyba! Nenalezen zdroj odkazů.**. základní stabilní stavy komunikačního kanálu) uložený v položce **CH_State**. Pomocí metody **ChState** je možno provádět test na dosažení základních stabilních stavů automatu komunikačního kanálu.

Tuto metodu další komunikační objekty zpravidla nepředefinovávají.

6.1.2.19. ChReady funkce

```
function  ChReady: TChState;
```

Metoda **ChReady** provede krok automatu komunikačního kanálu na základě volání metody **ChTick** a navrátí aktuální stav komunikačního kanálu uložený v položce **CH_Ctrl**. Také pomocí metody **ChReady** je možno provádět test na dosažení základních stabilních stavů automatu komunikačního kanálu.

Tuto metodu další komunikační objekty zpravidla nepředefinovávají.

Činnost komunikačního objektu je zpravidla implementována jako konečný automat. Voláním metod **ChState** nebo **ChReady** můžeme způsobit přechod automatu komunikačního kanálu do dalších jeho stavů. Přechod mezi dvěma sousedními stabilními stavy může být proveden přes několik různých nestabilních stavů, které jsou potřebné v daném komunikačním objektu.

6.1.2.20. ChDestNode procedura

```
procedure ChDestNode(Node: TNode);
```

Metoda **ChDestNode** určuje číslo (adresu) stanice u komunikací typu lokální síť, pro kterou bude zpráva určena. Následné volání metody **ChSend** způsobí zaslání zprávy stanici s adresou určenou parametrem `Node`.

6.1.2.21. ChSend procedura

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

Metoda **ChSend** způsobí započetí vysílání zprávy délky `Len` uložené na adrese určené ukazovatelem `Buff`. Po volání této metody by mělo následovat volání metody **ChSendReady** s testem na **CHS_SendReady** (čekací smyčka do odvysílání zprávy).

Příklad správného odvysílání zprávy metodou **ChSend**:

```
if ChSendReady = CHS_SendReady then {může se vysílat ?}
begin
  { naplnění bufferu daty }
  ...
  ChSend(Buff,Len); {započetí vysílání}
  repeat
    if ChSendResult<>res_Ok then ...; { dekódování chyby }
  until ChSendReady = CHS_SendReady;{čeká se do konce vysílání}
end
else
  writeln('Zatím nemohu vysílat');
```

Pokud aplikace používá operačního systému reálného času (o.s.Retos), měl by být v těle cyklu použit příkaz `Wait(1);` pro případné předání řízení i ostatním procesům.

6.1.2.22. 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**. Zpravidla se provádí pouze test na stabilní stav **CHS_SendReady**, protože ostatní stavy probíhajícího vysílání jsou nestabilní.

6.1.2.23. ChSendFlush procedura

```
procedure ChSendFlush;
```

Metoda **ChSendFlush** způsobí ukončení (přerušení) vysílání a přechod automatu vysílače do stavu **CHS_SendReady**.

6.1.2.24. ChReceiveBuffer procedura

```
procedure ChReceiveBuffer(Buff: pointer; Len: Word);
```

Voláním metody **ChReceiveBuffer** určíme adresu a velikost bufferu, do kterého se provede přesun přijaté zprávy. Parametrem `Buff` bývá zpravidla ukazatel na datový buffer definovaný aplikačním programem. Velikost bufferu je zadávána v počtu byte. Metodu **ChReceiveBuffer** je možno volat vícekrát a jednoduše měnit adresu bufferu pro uložení zprávy.

6.1.2.25. ChReceiveReady funkce

```
function  ChReceiveReady: TChState;
```

Metoda **ChReceiveReady** způsobí provedení kroku přijímacího automatu na základě volání metody **ChReceiveTick**. Jako svoji funkční hodnotu vrátí 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.

6.1.2.26. ChReceive procedura

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

Metoda **ChReceive** provede přijmutí zprávy a její uložení do přijímacího bufferu definovaném metodou **ChReceiveBuffer**. V proměnné `Len` navrací délku přijaté zprávy. Před voláním této metody, musí předcházet volání metody **ChReceiveReady** s testem na **CHS_ReceiveReady**, jinak v případě nepřijetí žádné zprávy skončí volání metody **ChReceive** chybou.

Příklad správného přijetí zprávy metodou **ChReceive**:

```
ChReceiveBuffer(MyBuff,MyLenBuff);  { nastavení přij. bufferu }
if ChReceiveResult<>res_Ok then ...;       { dekódování chyby }
```

```
while not ChReceiveReady=CHS_ReceiveReady do
begin                          { čekací smyčka pro příjem zprávy }
  if ChReceiveResult<>res_Ok then ...;     { dekódování chyby }
end;
ChReceive(..);                             { příjem zprávy }
if ChReceiveResult<>res_Ok then ...;       { dekódování chyby }
```

Pokud aplikace používá operačního systému reálného času (o.s.Retos), měl by být v těle cyklu použit příkaz `Wait(1);` pro případné předání řízení i ostatním procesům.

6.1.2.27. ChReceiveChar funkce

```
function  ChReceiveChar: Byte;
```

Metoda **ChReceiveChar** navrací jeden přijatý znak. Před jejím voláním musí předcházet volání metody **ChReceiveReady** s testem na **CHS_ReceiveReady**, jinak v případě nepřijetí žádného znaku skončí volání metody **ChReceiveChar** chybou. Komunikační objekty, které přijímají data ve formě celých zpráv, volají metodu **ChReceiveChar** následného komunikačního objektu, který byl zřetězen v komunikačním kanálu, prostřednictvím metody **ChReceive**. Sami metodu **ChReceiveChar** předefinovávají tak, aby její přímé volání skončilo chybou.

6.1.2.28. ChReceiveFlush procedura

```
procedure ChReceiveFlush;
```

Metoda **ChReceiveFlush** způsobí vyprázdnění přijímacích bufferů a nastavení stavu přijímače kanálu **CH_RCtrl** na stabilní stav **CHS_ReceiveNoReady** nebo na jiný stabilní stav. Komunikační objekty nižších vrstev nastavují položku **CH_RCtrl** zpravidla na stav **CHS_ReceiveNoReady**, kdežto komunikační objekty vyšších vrstev ji nastavují na své vlastní stabilní stavy.

6.1.2.29. 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. Je-li adresa příjemce (DNode) rovna 0, pak přijatá zpráva byla určena všem připojeným stanicím. Tuto metodu má smysl volat po přijetí zprávy metodou **ChReceive**.

6.1.2.30. ChResult funkce

```
function  ChResult: tChResult;
```

Metodou **ChResult** je navracena hodnota položky **CH_Result**, ve které je v případě chyby uložen výsledek první prováděné operace nad komunikačním kanálem, která skončila chybou. Výsledkem je dvoubajtová hodnota, kde v horním byte je uloženo číselné jméno knihovny (CH_NumName), která zjistila chybu, a v nižším byte je číselný kód chyby. Pokud nenastala žádná chyba, je v položce **CH_Result** hodnota **res_Ok**. Po zavolání této metody se položka **CH_Result** vynuluje (nastaví na hodnotu **res_Ok**). Ostatní metody by měly položku **CH_Result** nastavovat pomocí metody **ChSetResult**.

6.1.2.31. ChSendResult funkce

```
function  ChSendResult: tChResult;
```

Metodou **ChSendResult** je navracena hodnota položky **CH_SResult**, ve které je v případě chyby uložen výsledek první prováděné operace nad vysílačem komunikačního kanálu, která skončila chybou. Pokud nenastala žádná chyba, je v položce **CH_SResult** hodnota **res_Ok**.

Metoda **ChSendResult** a formát hodnoty položky **CH_SResult** mají stejné vlastnosti jako metoda **ChResult** a položka **CH_Result**. Ostatní metody by měly položku **CH_SResult** nastavovat pomocí metody **ChSetSendResult**.

6.1.2.32. ChReceiveResult funkce

```
function  ChReceiveResult: tChResult;
```

Metodou **ChReceiveResult** je navracena hodnota položky **CH_RResult**, ve které je v případě chyby uložen výsledek první prováděné operace nad přijímačem komunikačního kanálu, která skončila chybou. Pokud nenastala žádná chyba, je v položce **CH_RResult** hodnota **res_Ok**.

Metoda **ChReceiveResult** a formát hodnoty položky **CH_RResult** mají stejné vlastnosti jako metoda **ChResult** a položka **CH_Result**. Ostatní metody by měly položku **CH_RResult** nastavovat pomocí metody **ChSetReceiveResult**.

6.1.2.33. ChSetResult procedura

```
procedure ChSetResult(Value: tChResult);
```

Metoda **ChSetResult** nastaví položku **CH_Result** na zadanou hodnotu v případě, že položka dosud neobsahuje kód chyby, ale hodnotu **res_Ok**. Tím je zabezpečeno, že v položce **CH_Result** zůstane uložena případná chyba a nebude docházet k jejímu přemazávání.

Tuto metodu smí volat pouze metody komunikačních objektů, nesmí ji však předefinovat. Aplikace tuto metodu nesmí volat přímo (je prostřednictvím některé z metod komunikačního objektu).

6.1.2.34. ChSetReceiveResult procedura

```
procedure ChSetReceiveResult(Value: tChResult);
```

Metoda **ChSetReceiveResult** nastaví položku **CH_RResult** na zadanou hodnotu v případě, že položka dosud neobsahuje kód chyby, ale hodnotu **res_Ok**. Tím je zabezpečeno, že v položce **CH_RResult** zůstane uložena případná chyba a nebude docházet k jejímu přemazávání. Metoda je podobná metodě **ChSetResult** a platí pro ni stejná pravidla.

6.1.2.35. ChSetSendResult procedura

```
procedure ChSetSendResult(Value: tChResult);
```

Metoda **ChSetSendResult** nastaví položku **CH_SResult** na zadanou hodnotu v případě, že položka dosud neobsahuje kód chyby, ale hodnotu **res_Ok**. Tím je zabezpečeno, že v položce **CH_SResult** zůstane uložena případná chyba a nebude

docházet k jejímu přemazávání. Metoda je podobná metodě **ChSetResult** a platí pro ni stejná pravidla.

6.1.2.36. ChTick procedura

```
procedure ChTick;
```

Metoda **ChTick** způsobí provedení jednoho či více kroků automatu komunikačního kanálu. Je nutné ji periodicky volat. Metoda **ChTick** je rovněž automaticky volána v metodách **ChReady** a **ChState**.

Ostatní komunikační objekty tuto metodu zpravidla předefinovávají.

6.1.2.37. ChSendTick procedura

```
procedure ChSendTick;
```

Metoda **ChSendTick** způsobí provedení jednoho či více kroků automatu vysílače. Při vysílání je nutné ji periodicky volat. Metoda **ChSendTick** je rovněž automaticky volána v metodě **ChSendReady**.

Ostatní komunikační objekty tuto metodu zpravidla předefinovávají.

6.1.2.38. ChReceiveTick procedura

```
procedure ChReceiveTick;
```

Metoda **ChReceiveTick** způsobí provedení jednoho či více kroků automatu přijímače. U vyšších knihoven při přijímání je nutné ji periodicky volat. Metoda **ChReceiveTick** je rovněž automaticky volána v metodě **ChReceiveReady**.

Ostatní komunikační objekty tuto metodu zpravidla předefinovávají.

6.2. tAddChnVirt

Typ **tAddChnVirt** je typem objektu, který slouží k definování prvku v seznamu správců komunikačních objektů **ChnCollection**. Objekt správce komunikačního objektu je schopen vygenerovat instanci daného komunikačního objektu. Ostatní komunikační knihovny vytvářejí od tohoto typu dědice (tAddChnXxx) a předefinovávají pouze metodu **ChInit**.

6.2.1. Položky

`ChType : TChName;`

Položka definující jméno typu komunikačního objektu.

`ChName : TChName;`

Položka definující uživatelské jméno komunikačního objektu.

`ChNumName : TChNumName;`

Položka definující číselné jméno typu komunikačního objektu.

`Channel : PChnVirt;`

Položka definující ukazatel na již vytvořenou instanci komunikačního objektu.

6.2.2. Metody

6.2.2.1. Init konstruktor

`constructor Init (Chn: PChnVirt);`

Konstruktor **Init** slouží k vytvoření instance objektu správce daného komunikačního objektu.

6.2.2.2. ChInit funkce

`function ChInit: PChnVirt;`

Metoda **ChInit** slouží k vytvoření instance daného komunikačního objektu a ukazatel na tento objekt vrací jako svoji funkční hodnotu. Ostatní objekty dědičné od rodičovského objektu tAddChnVirt musí tuto metodu předefinovat na základě vlastního spravovaného komunikačního objektu.

6.3. tChnCollection

Typ **tChnCollection** je typem objektu, který slouží k vytvoření a udržování seznamu objektů správců komunikačních objektů (*seznamu správců*). Objekt tChnCollection je rovněž (stejně jako objekt správce komunikačního objektu) schopen vygenerovat instanci daného komunikačního objektu.

6.3.1. Metody

6.3.1.1. Init konstruktor

```
constructor Init;
```

Konstruktor **Init** slouží k vytvoření seznamu správců komunikačních objektů.

6.3.1.2. Insert procedura

```
procedure Insert(Item: Pointer);
```

Metoda **Insert** slouží ke vložení prvku správce komunikačního objektu do seznamu správců. Tato metoda by se měla volat v implementační části každé komunikační knihovny.

6.3.1.3. ChInsert procedura

```
procedure ChInsert(Chn: pChnVirt);
```

Metoda **ChInsert** vloží odkaz na instanci daného komunikačního objektu do seznamu správců.

6.3.1.4. ChDelete procedura

```
procedure ChDelete(Chn: pChnVirt);
```

Metoda **ChDelete** zruší odkaz na instanci daného komunikačního objektu v seznamu správců.

6.3.1.5. ChNewInit funkce

```
function  ChNewInit(Name: tChName): pChnVirt;
```

Metoda **ChNewInit** vytvoří instanci daného komunikačního objektu ze seznamu správců a zavolá konstruktor **Init** daného komunikačního objektu. Ukazatel na instanci vzniklého komunikačního objektu vrátí jako svoji funkční hodnotu.

6.3.1.6. ChNumName funkce

```
function  ChNumName(Name: tChName): tChNumName;
```

Metoda **ChNumName** nalezne v seznamu správců prvního správce, který spravuje komunikační objekt s daným textovým jménem a vrátí jeho číselné jméno.

6.3.1.7. ChName funkce

```
function  ChName(NumName: tChNumName): tChName;
```

Metoda **ChName** nalezne v seznamu správců prvního správce, který spravuje komunikační objekt s daným číselným jménem a vrátí jeho textové jméno.

6.3.1.8. ChAllName funkce

```
function  ChAllName   :String;
```

Metoda **ChAllName** vrátí řetězec textových jmen všech komunikačních objektů, které jsou spravovány pomocí seznamu správců.

6.3.1.9. ChAllNumName funkce

```
function  ChAllNumName:String;
```

Metoda **ChAllNumName** vrátí řetězec textových a číselných jmen všech komunikačních objektů, které jsou spravovány pomocí seznamu správců.

7. Procedury a funkce

7.1. ChnVirt_ResultToStr funkce

```
function  ChnVirt_ResultToStr(Res:tChResult):tResultStr;
```

Funkce **ChnVirt_ResultToStr** převede kód výsledku v parametru Res na textový řetězec. Funkce převádí kódy uvedené v kapitole „4.2 Konstanty výsledků operací“. Každá další knihovna **ChnXxx** definuje vlastní funkci **ChnXxx_ResultToStr**, která převádí své specifické výsledky operací.

V praxi bývá komunikační kanál tvořen více vrstvami (například protokolová vrstva tvořená knihovnou ChnPrt a pod ní fyzická vrstva tvořená knihovnou ChnCom). Máme-li instanci objektu na takto zřetězený komunikační kanál, a některá z metod ChResult, ChReceiveResult nebo ChSendResult nám vrátí kód výsledku operace, použijeme následující algoritmus jako příklad pro vypsání textového řetězce výsledku operace:

Např:

```
uses
  NumToStr,
  ChnVirt,
  ChnPrt,
  ChnCom;

var
  Res : tChResult;
  MyCh: pChnVirt;

begin
  ...
  {vytvoření instance MyChn a nastavení parametrů pro vrstvy
    ChnPrt a ChnCom}
  {zavoláme některou metodu objektu např. ChOpen}
  MyChn^.ChOpen;
  Res:= MyChn^.ChResult; {zjištění a uložení výsledku operace
                          ChOpen}
```

```
      if Res<>res_Ok then
        case Res and $FF00 of
          ChNumName('PRT'): write('PRT:', ChnPrt_ResultToStr(Res));
          ChNumName('COM'): write('COM:', ChnCom_ResultToStr(Res));
          else              write('Err:$',WordStrHex(Res));
        end;
```

8. Příklad

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

```
program ExampleChnVirt;
uses
  uString,
  ChnVirt,
  ChnCom,
  ChnPrt,
  ...
const
  {parametry komunikačního kanálu}
  ParamStr : tParamStr =
             'NAM=PRT LSB=5000 NOD=1 DNO=2 '+
             'NAM=COM COM=1 IRQ=4 BD=1200 BIT=8 STOP=2 LRB=1000';

type
  { typ visílacího a přijímacího bufferu - jenom pro příklad,
    uživatel si může nadefinovat typ bufferu dle svých potřeb }
  pMess    = ^tMess;
  tMess    = array [0..20] of Byte;

const
  Chn      : pChnVirt = nil; {ukazatel na instanci komunikačního
                              objektu}
  SMess    : pMess    = nil; {ukazatel na vysílací buffer}
  RMess    : pMess    = nil; {ukazatel na přijímací buffer}
  LSMess   : Word     = 0;   {délka vysílané zprávy}
  LRMess   : Word     = 0;   {délka přijaté zprávy}

begin
  ...
  { inicializace přijímacího a vysílacího bufferu }
  New(SMess);
  New(RMess);
  ...
  { vytvoření instance Chn }
  Chn:=ChnCollection^.ChNewInit(ChnPRT.cName);
  with Chn^ do
  begin
    { nastavení parametrů komunikace }
    ChSetParam(ParamStr);
    if ChResult<>res_Ok then WriteLn('Chyba');
    { Open kanálu }
    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 přijatá zpráva uložit }
    ChReceiveBuffer(RMess,SizeOf(tMess));
    if ChReceiveResult<>res_Ok then WriteLn('Chyba');
```

```
    { Connect kanálu }
    ChConnect;
    repeat
      if ChResult<>res_Ok then WriteLn('Chyba');
    until ChReady=CHS_Connect;
    if ChResult<>res_Ok then WriteLn('Chyba');
    ...
    { nadefinování zprávy uživatelem }
    SMess^:=...;
    Lmess :=...;
    { vyslání zprávy }
    if ChSendReady=CHS_SendReady then
    begin
      ChSend(SMess, LMess);
      { 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');
    ...
    { 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); Chn := nil;
  Dispose(SMess); SMess := nil;
  Dispose(RMess); RMess := nil;
  ...
end.
```