KitKing

POPIS VIZUALIZAČNÍCH KNIHOVEN PRO ŘÍDICÍ SYSTÉM KIT

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 1.11

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: 16.05.2003
Datum posledního uložení dokumentu: 16.05.2003
(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	verze do 17.12.2002	Hv	19.12.2002	První vydání
1.10	2.XX	Hv	5.2.2003	Zavedení jednotné verze programového balíku v jednotce WTVirt pod identifikátorem cVer (smazány dosavadní cName a cVer). Oprava ukázky kódu v 4.4
1.11	2.XX	Tu	16.05.2003	Úprava dokumentu dle ISO9000. Opravená hlavička fce fnGetListExItem. Doplněné konstanty Res_ErrChannelNotCreate a Res_ErrAllocateMem.

1.2. Účel dokumentu

Tento dokument slouží jako popis programového balíku knihoven KitKing používaného jako protistanice programu TheKing.

1.3. Rozsah platnosti

Určen pro programátory a uživatele programového vybavení SofCon.

1.4. Související dokumenty

Pro čtení tohoto dokumentu není potřeba číst žádný další manuál, ale je potřeba orientovat se v používání programového vybavení SofCon.

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

Balík knihoven je určen pro řídicí systémy KIT a slouží jako protistanice programu TheKing. Protože funkce knihoven vychází z možností poskytovaných tímto programem, bude v následující části jeho krátký popis.

Program TheKing lze spustit na počítači s operačním systémem Windows 98 a novější (Windows XP) a nabízí Vám tyto možnosti:

- zobrazení archivů (pole datových struktur). Stačí pouze několik kliknutí myši a data můžete zobrazit pomocí tabulky nebo grafů. Zobrazovaná data můžou být on-line s volitelnou rychlostí aktualizace nebo off-line z vybraného souboru.
  Archivy jsou především používány pro opakované ukládání dat se zadanou periodou.
- přenos tzv. on-line parametrů řídicího systému – lze přenášet jednotlivě i blokově
- přenos binárních dat
- nahrávání nové verze firmware do řídicího systému
- uživatelská definice šablon pro vykreslování grafů
- jednoduchý editor uživatelských obrazovek
- možnost používání přístupových práv. Pomocí těchto práv můžete omezovat nejen jednotlivé prvky (tlačítka, editační pole apod.), ale i zobrazování celého okna
- knihovny pro podporu vývoje aplikace

Balík knihoven poskytuje programátorovi jednoduché rozhraní v jazyce Borland Pascal. Přestože při jeho implementaci byly použity objekty pro zapouzdření komunikačních protokolů a automatů, nemusí programátor znát objektové programování.

Rozšíření stávajících aplikací o vizualizaci je max. zjednodušeno (viz dodávané příklady) a při běžném používání spočívá pouze v doplňování výkonné části. Při speciálních požadavcích lze s výhodou využít možností objektové stavebnice.

4. Postup pro přidání vizualizace do stávající aplikace

Pokud vytváříte svoji první vizualizaci doporučujeme Vám přečíst manuál Začínáme vizualizovat.

4.1. Komunikační knihovny

Protože komunikační knihovny při svém uvedení v části **uses** provádí registraci ve svém správci, musí se podle zvoleného způsobu přenosu dat tato část upravit.

V případě, že aplikace už tyto komunikační knihovny používá tzn. má je uvedeny v jiných jednotkách lze přeskočit na bod 4.2.

Pozn. Úprava **uses** platí pro komunikační objekty odvozené od ChnVirt tak od CoBase.

```
uses
  { .. }
{ komunikacni knihovny odpovidajici prenosu dat }
  ;
```

4.2. ReTOS

Protože komunikace s nadřízeným počítačem (vizualizací) probíhá ve vyhrazeném procesu s nastavitelnou prioritou, musí aplikace používat operační systém ReTOS. V případě, že aplikace už ReTOS používá, lze přeskočit na bod 4.3.

Pokud aplikace ReTOS nepoužívá musí se aplikace rozšířit o následující kód.

```
program Priklad
uses
  { .. }
  , Kernel,    { jadro os ReTOS }
{ komunikacni knihovny, viz predchozi bod }
  ;
{ kod programu }
begin
{ kod programu }
{ po inicializaci struktur a promennych programu, vcetne
promennych s rozlozenim pameti se provede inicializace
ReTOS a spusteni hlavniho procesu }
  StartMain(200, 254);
  InitInterruptStack(1,254);
  StartTimeSlicing(8);
{ kod programu }
end.
```

4.3. Vytvoření komunikačního objektu

Protože ve většině případů komunikace běží po celou dobu řízení lze použít statický objekt. Jeho inicializace a vytvoření se provede následujícím kódem.

```
var
  TheWinCom:TWinCom;    { komunikacni objekt spravujici
       jednotlive objekty vizualizace - soubory, archivy,
       prenos online parametru a aplikaci LOADER }
begin
  { po inicializaci programu a ReTOS }
  { inicializace komunikacniho kanalu }
  TheWinCom.Init;
{ kod programu }
end.
```

4.4. Vytvoření komunikačního kanálu a nastavení jeho parametrů

Počet kroků potřebných pro vytvoření celého komunikačního kanálu závisí na zvoleném přenosu dat.

Pokud se nepoužívají komunikační objekty odvozené od CoBase (např. přenos dat pomocí UDP), je celý komunikační kanál vytvořen po zavolání metody CreateChn. Jinak řečeno tato metoda vytvoří řetězec komunikačních objektů odvozených od ChnVirt. Po jejich vytvoření je možné nastavit parametry pro objekty odvozené od CoBase (např. v objektu UDPPRT). Objekty odvozené CoBase se vytvoří až po zavolání metody Open.

```
{ po inicializaci programu, ReTOS a objektu typu TWinCom}
  { vytvoreni retezce kom. objektu odvozenych od ChnVirt}
  if TheWinCom.CreateChn(strComm)<>Res_Ok then
  begin
    { obsluha chyby. Zde doporucujeme vypsat zpracovani
       parametrizacniho retezce, viz
       TheWinCom.m_ChnDesc.pChannel^.ChnGetParam('') }
  end;
  { nastaveni parametru pro objekty odvozene od CoBase }
  { dokonceni vytvoreni kom. kanalu }
  if TheWinCom.Open<>Res_Ok then
  begin
    { obsluha chyby. Zde doporucujeme vypsat zpracovani
       parametrizacniho retezce, viz
       TheWinCom.m_ChnDesc.pChannel^.ChnGetParam('') }
  end;
```

4.5. Přidání vybrané vizualizační funkce do aplikace

Tento bod se provádí ve dvou krocích. Nejprve se provede vytvoření potomka objektu definujícího vybranou funkci vizualizace a poté se tento objekt přidá do komunikačního objektu. Protože vytvoření potomka objektu a doplnění výkonných částí bude věnována následující kapitola, budeme předpokládat, že v programu je definován objekt s přenosem online parametrů pojmenovaný PParam (=^TParam). Rozšíření komunikačního objektu o funkci přenosu parametrů se provede:

```
var
  pTheParam:PParam;

begin
{ inicializace programu, ReTOS a komunikacniho kanalu }
  pTheParam=new(PParam, Init);
  if not TheWinCom.Register(pTheParam) then
  begin
    { obsluha chyby }
  end;

{ kod programu }

end.
```

Výše uvedený kód provádí následující. Nejdříve vytvoří a inicializuje objekt s výkonnými částmi objektu. Poté následuje připojení objektu přenosu parametrů do komunikačního objektu, tzn. od této chvíle se objekt přenosu parametrů podílí na zpracování přijatých zpráv. Podrobný popis bude uveden v kapitole věnované komunikačnímu objektu.

4.6. Spuštění komunikace

Spuštění komunikace se provede dále uvedeným kódem. V rámci tohoto kódu se vytvoří samostatný proces se jménem WinCom pomocí volání TheWinCom.CreateProcess. Podrobnější popis bude uveden v kapitole věnované komunikačnímu objektu.

```
begin
{ inicializace programu, ReTOS, komunikacniho kanalu,
objektu s vybranou vizualizacni funkci a jeho pripojeni
ke komunikacnimu objektu }

  { spusteni komunikacniho procesu }
  if not TheWinCom.CreateProcess then
  begin
    { obsluha chyby }
  end;

{ kod programu }

end.
```

Po vytvoření samostatného procesu komunikace je Vaše aplikace schopná vykonávat požadované funkce vizualizace. Tyto funkce je možné okamžitě ověřit pomocí programu TheKing.

4.7. Ukončení komunikace

Ukončení této komunikace se provede následujícím kódem.

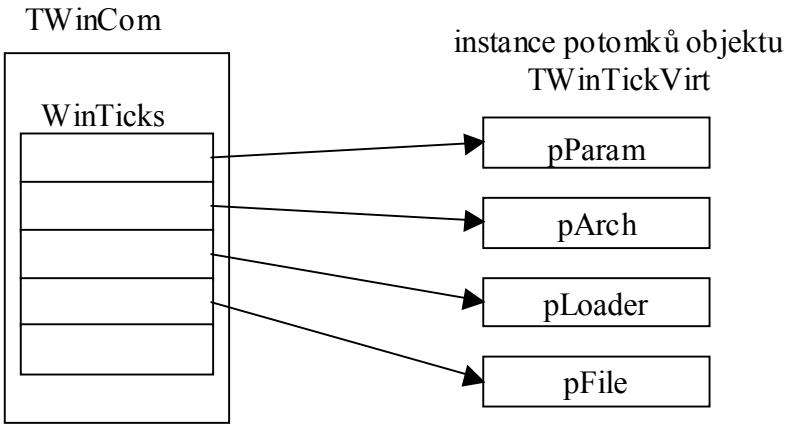
```
begin
{ kod programu provadejici jeho ukonceni }
  { odpojeni objektu vizualizace }
  if not TheWinCom.UnRegister(pTheParam) then
  begin
    { obsluha chyby }
  end;
  Dispose(pTheParam, Done);
  if not TheWinCom.DestroyProcess then
  begin
    { obsluha chyby }
  end;
  { uzavreni komunikacniho kanalu }
  TheWinCom.Close;
  { zruseni retezce komunikacnich objektu }
  TheWinCom.DestroyChn;
{ kod programu }
end.
```

5. Komunikační knihovna WinCom

5.1. Úvod

Komunikační objekt TWinCom slouží pro zastřešení obecného protokolu SofConL2. Nad tímto protokolem jsou vytvořeny komunikační automaty pro přenos dat archivů, souborů, apod. Protože jejich implementace je nezávislá od vlastního protokolu, jsou objekty s automaty odvozovány od objektu TWinTickVirt, který nemá vazbu (tj. není potomkem) s objektem TWinCom. Objekt TWinTickVirt implementuje několik základních virtuálních metod, které jsou určeny k předefinování v potomcích objektu. Pro tuto chvíli je důležité vědět, že objekt umí zpracovat každou přijatou zprávu a na ní vracet chybový kód.

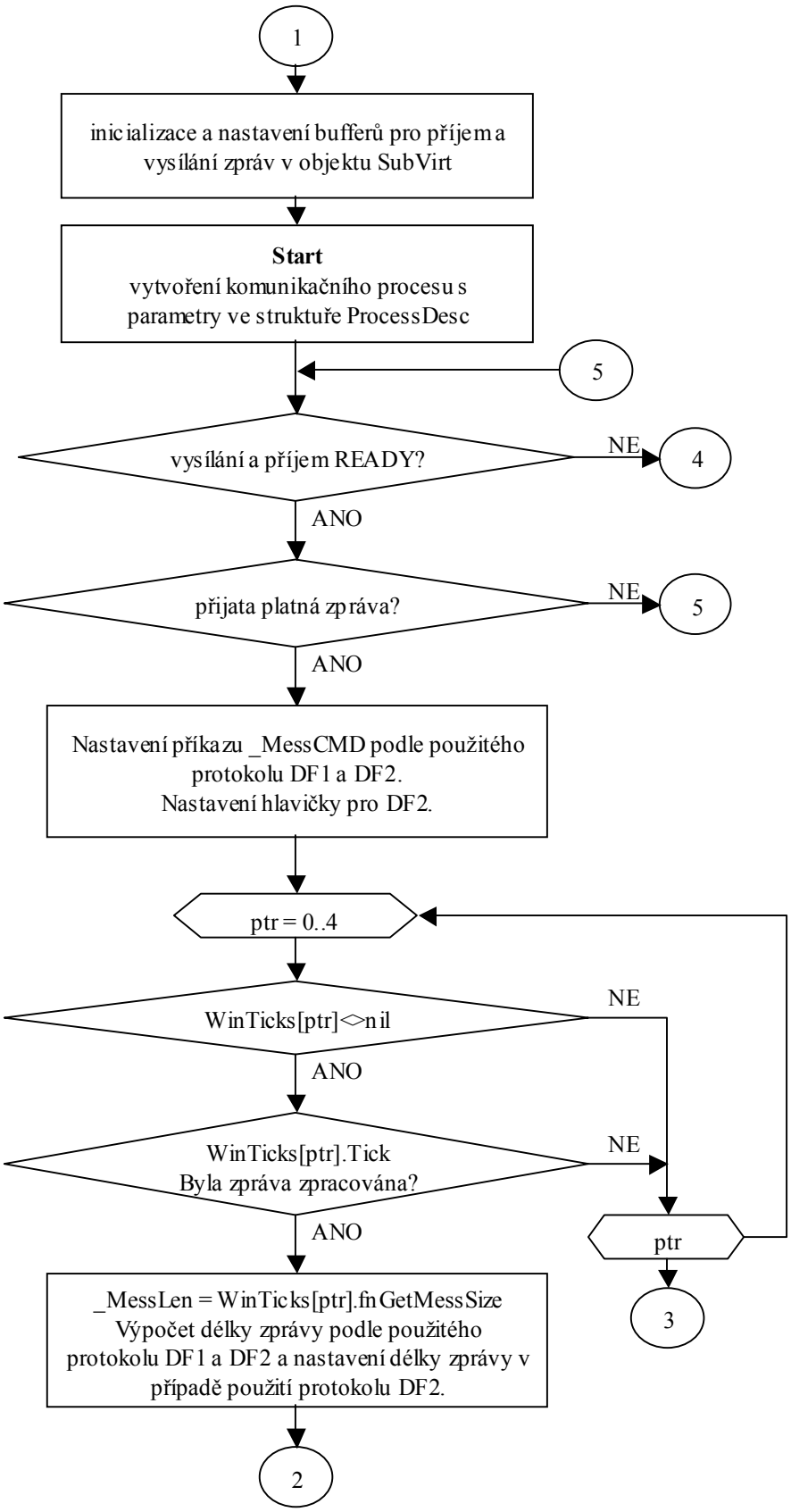
Vztah komunikačního objektu TWinCom a instancí potomků objektu TWinTickVirt je zachycen na následujícím obrázku.

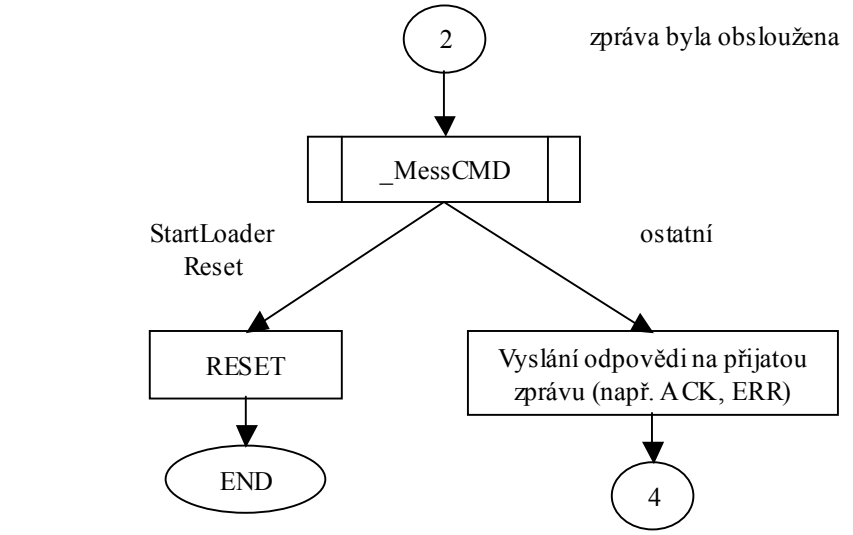


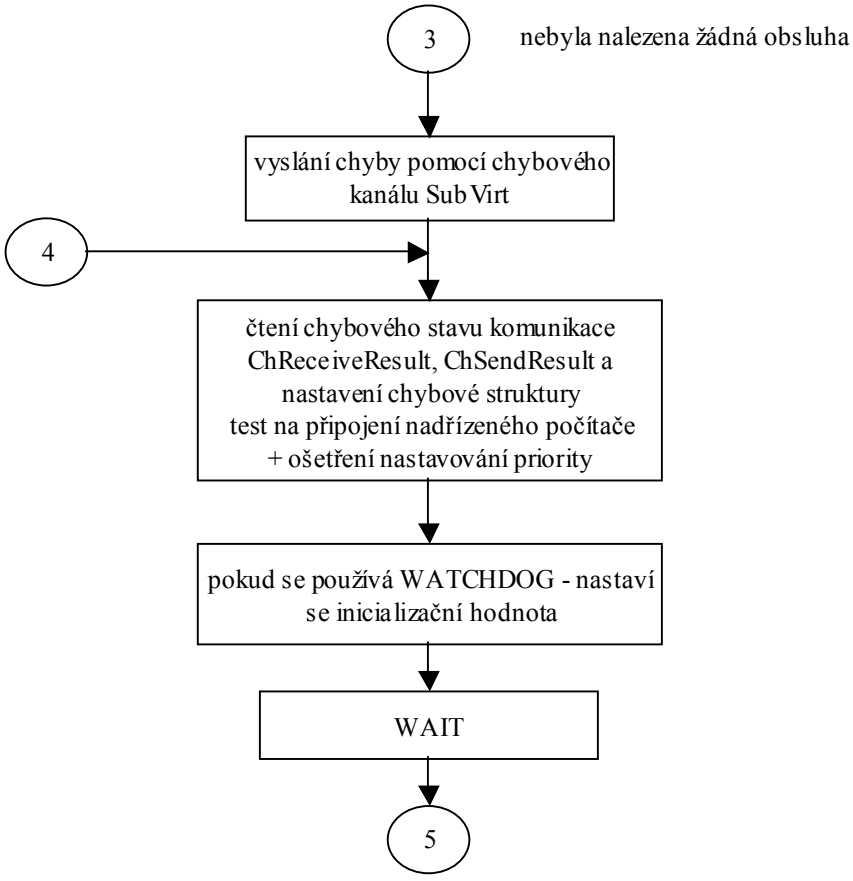
odkazy na instance objektů, které implementují
jednotlivé funkce (dědici TWinTickVirt)

Po inicializaci objektu TWinCom se provede přidání funkčních objektů do vlastní obsluhy zpracování zpráv. Pokud se přidání neprovede, vrátí komunikační objekt chybový kód. Vlastní přidání se provede voláním funkce Register, které se jako parametr předá ukazatel na funkční objekt. Ten je potomkem objektu TWinTickVirt nebo přímo potomkem objektů implementujících komunikační automaty jednotlivých funkcí. (Pozn. Objekt TWinCom umožňuje registraci pouze jedné instance jednoho typu funkčního objektu. Jednotlivé funkční objekty by proto v aplikaci měly být použity ve významu správců, tj. všech archívů v aplikaci apod.) Vlastní zpracování odpovídá koncepci komunikačních knihoven odvozených od objektu ChnVirt, viz příslušný manuál. Protože programátor implementující nějakou funkci vizualizace nemusí znát přesná pravidla používání tohoto objektu, budou v další části uváděny jenom základní pravidla potřebná pro vlastní komunikaci. Další pravidla může programátor vyhledat v příslušných manuálech komunikačních knihoven.

Vlastní algoritmus zpracování přijaté zprávy probíhá podle následujícího vývojového diagramu:


1
inicializace a nastavení bufferů pro příjem a vysílání zpráv v objektu SubVirt
Start
vytvoření komunikačního procesu s parametry ve struktuře ProcessDesc
5
vysílání a příjem READY?
NE
4
ANO
přijata platná zpráva?
NE
5
ANO
Nastavení příkazu _MessCMD podle použitého protokolu DF1 a DF2.
Nastavení hlavičky pro DF2.
ptr = 0..4
WinTicks[ptr]<>nil
NE
ANO
WinTicks[ptr].Tick
Byla zpráva zpracována?
NE
ANO
ptr
3
_MessLen = WinTicks[ptr].fnGetMessSize
Výpočet délky zprávy podle použitého protokolu DF1 a DF2 a nastavení délky zprávy v případě použití protokolu DF2.
2


2
zpráva byla obsloužena
_MessCMD
StartLoader
Reset
ostatní
RESET
Vyslání odpovědi na přijatou zprávu (např. ACK, ERR)
END
4


3
nebyla nalezena žádná obsluha
vyslání chyby pomocí chybového kanálu SubVirt
4
čtení chybového stavu komunikace ChReceiveResult, ChSendResult a nastavení chybové struktury
test na připojení nadřízeného počítače
+ ošetření nastavování priority
pokud se používá WATCHDOG - nastaví se inicializační hodnota
WAIT
5

V diagramu je vidět používání členské proměnné WinTicks, pole objektů potomků TWinTickVirt implementujících jednotlivé funkce vizualizace. Nastavení a zrušení položek tohoto pole se provádí pomocí funkcí Register a UnRegister, jejichž parametrem je ukazatel na potomka objektu TWinTickVirt. Vlastní zpracování přijatých zpráv se provádí v těle metody Tick, jejíž návratová hodnota signalizuje zpracování či nezpracování přijaté zprávy. Pokud byla zpráva obsloužena, provádí se v dalším kroku výpočet délky celé zprávy následované jejím zařazením do vysílacího bufferu. Protože komunikační automaty neumí obsluhovat zpracování několika požadavků na vysílání, provádí se obsluha nové zprávy až po úplném odvysílání zprávy v komunikačním bufferu. Proto je ve vývojovém diagramu test na stav vysílacího a přijímacího kanálu.

5.2. Konstanty

```
cMaxWinTick = 5
```

Konstanta udává maximální počet funkčních objektů používaných při zpracování přijaté zprávy.

```
Res_ErrAllocateMem      = $00A0; {nelze alokovat paměť}
Res_ErrChannelNotCreate = $00A1; {volání open bez vytvořeného kanálu}
```

Kódy chybových výsledků metod objektu.

5.3. Typy

```
PMsgCntDesc = TMsgCntDesc;
TMsgCntDesc = record
 RecCnt     : LongInt;       { pocitadlo prijatych zprav }
 SendCnt    : LongInt;       { pocitadlo odvysilanych zprav }
 RecErrCnt  : LongInt;       { pocitadlo chyb pri prijmu zprav }
 SendErrCnt : LongInt;       { pocitadlo chyb pri vysilani zprav }
end;
```

Pokud položka komunikačního objektu pMsgCnt ukazuje na proměnnou tohoto typu, používá komunikační proces tato počítadla.

```
TProcessDesc = record        { vlastnosti komunikacniho procesu }
 Name       : tNameOfProces; { jmeno procesu )
 Stack      : Word;          { velikost zasobniku }
 SPrio      : Word;          { staticka priorita v klidovem stavu }
 SPrioF     : Word;          { staticka priorita pri prenosu dat }
 DPrio      : Word;          { dynamicka priorita }
end;
```

Struktura slouží pro určení vlastností komunikačního procesu. Při vytváření komunikačního procesu se použije položka ProcessDesc. Její standardní nastavení bude uvedeno u popisu komunikačního objektu.

```
TChannelDesc = record          { vlastnosti komunikacniho kanalu }
 pSMess     : Pointer;         { ukazatel na zpravu urcenou k vysilani,
                                 zprava muze byt prenasena protokolem DF1
                                 nebo DF2 }
 pRMess     : Pointer;         { ukazatel na prijatou zpravum, zprava
                                 muze byt prenasena protokolem DF1 nebo
                                 DF2 }
 pChannel   : pChnVirt;        { ukazatel na instanci komunikacniho
                                 objektu, ktery je potomkem pChnVirt }
 wSendingTime                  : Word; { doba pro okamzite vyslani
                                 zpravy }
 wPcConnectTime                : Word; { timeout pro ztratu komunikace s
                                 nadrizenym PC }
 wReadyTime : Word;            { timeout pro osetreni chybovych stavu
                                 pri otevirani, pripojovani nebo
                                 zavirani, odpojovani komunikace }
 fRaisePrio : Boolean;         { pri navazani komunikace s nadrizenym
                                 pocitacem se muze staticka priorita
                                 komunikacniho procesu zvysit na hodnotu
                                 SPrioF }
 fFast      : Boolean;         { priznak, ze komunikacni proces je
                                 spousten s vyssi prioritou }
 fPcConnected                  : Boolean;    { priznak, ze komunikacni
                                 proces je pripojen k nadrizenemu
                                 pocitaci. Pri zmene tohoto priznaku se
                                 dle nastaveni fFast meni priorita
                                 komunikacniho procesu }
 theCommTimer                  : tTimer;     { pomocny casovac pro
                                 mereni doby pri praci s komunikacni
                                 kanalem }
end;
```

Struktura popisující stav a nastavení komunikačního kanálu. Při otevírání a připojování komunikačního kanálu se použije položka ChnDesc. Podrobnější popis bude uveden u popisu komunikačního objektu.

5.4. Objekt TWinCom

5.4.1. Položky

```
 m_WinTicks                    : array [0..cMaxWinTicks-1] of
PWinTickVirt;
```

Pole funkční objektů zpracovávajících přijaté zprávy. Položky tohoto pole jsou v komunikačním procesu použity pro zpracování přijaté zprávy.

```
 m_ProcessDesc                 : TProcessDesc;
```

Položka popisuje vlastnosti komunikačního procesu, který se vytvoří voláním metody CreateProcess.

```
 m_ChnDesc  : TChannelDesc;
```

Položka popisuje vlastnosti a stav komunikačního kanálu, který se vytvoří voláním metody Open.

```
 m_pMsgCnt  : PMsgCnt;
```

Ukazatel na strukturu, která obsahuje počítadla přijatých a odvysílaných zpráv, případně počítadla chyb při příjmu a vysílání zpráv.

```
m_fUseDF2  : Boolean;
```

Při přenosu dat se používá protokol DF2, který navíc ve zprávě přenáší hlavičku obsahující informace o vysílaném procesu v programu TheKing. Hodnota této položky se nastaví po zavolání metody Open na hodnotu nastavenou v přenosovém kanálu. Její hodnotu lze změnit pouze po zavření komunikačního kanálu a jeho novém otevření, protože její hodnota určuje jak je přijatá a vysílaná zpráva zpracovávána – protokol DF1 a DF2.

```
m_wInitWDCnt                : Word;
```

Inicializační hodnota počítadla pomocí něhož se sleduje běh komunikačního procesu – tzv. WatchDog.

```
m_pWDCnt   : ^Word;
```

Ukazatel na počítadlo pomocí něhož se sleduje běh komunikačního procesu – tzv. WatchDog. Počítadlo pWDCnt^ je nastaveno na inicializační hodnotu m_wInitWDCnt v každém cyklu komunikačního procesu a toto počítadlo by mělo být obsluhováno v přerušovací rutině IRQ0. Např. pomocí jednotky Tick a procedury UserTick1 a UserTick2.

5.4.2. Metody

5.4.2.1. Konstruktor Init

```
constructor Init;
```

Konstruktor slouží k dokončení inicializace objektu s virtuálními metodami a jeho položek. Položky objektu jsou inicializovány následovně.

Položky pole m_WinTicks jsou nastaveny na NIL.

m_pMsgCnt, m_pWDCnt = NIL, m_wInitWDCnt = 0 a m_fUseDF2=TRUE.

Položky m_ProcessDesc jsou nastaveny následovně:

Name = WinCom
Stack = 8000
SPrio = 90
SPrioF = 200
DPrio = 254

Položky m_ChnDesc jsou nastaveny následovně:

pSMess, pRMess, pChannel = NIL;
wSendingTime = 1500;
wPcConnecTime = 1500;
wReadyTime = 1500;
fRaisePrio = False;
fFast = False;
fPcConnected = False;
Dále se volá konstruktor instance objektu theCommTimer.

5.4.2.2. Destruktor Done

```
destructor Done; virtual;
```

Destruktor slouží ke zrušení komunikačního objektu. Před voláním této funkce musí uživatel zajistit zrušení komunikačního procesu a jednotlivých funkčních objektů.

5.4.2.3. ImmediateSendMess

```
procedure ImmediateSendMess; virtual;
```

Metoda pro okamžité odvysílání zprávy připravené ve vysílacím bufferu. Odvysílání se provádí max. po dobu definovanou položkou objektu m_ChnDesc.wSendingTime, pokud se po tuto dobu odvysílání nepodaří, tak se pokračuje dále – zpravidla se vyvolá RESET.

V každém cyklu vysílání zprávy se nastavuje počítadlo pro sledování běhu komunikačního procesu na inicializační hodnotu m_wInitWDCnt.

5.4.2.4. CreateChn

```
function CreateChn(const strComm:String):tChResult;
```

V této funkci se provedou alokace bufferu pro příjem a vysílání a vytvoření řetězce komunikačních objektů odvozených od ChnVirt podle předaného konfiguračního parametru. Pravidla pro sestavení konfiguračního parametru jsou popsány v manuálech použitých komunikačních protokolů a knihoven.

Po zavolání této metody lze měnit parametry jednotlivých objektů pomocí konfiguračního řetězce a použití metody ChSetParam, případně přímo nastavovat parametry určené pro komunikační objekty odvozené od CoBase. Tyto objekty se totiž vytváří až při volání metody Open.

Při přenosu dat pomocí protokolu SofConL2 přes standardní komunikační port by konfigurační řetězec vypadal následovně. (Pozn. Popis jednotlivých parametrů najdete v manuálu ChnSofs2, ChnPrt, ChnCom případně ChnVirt.)

strComm = 'NAM=SOFS2 LRB=2500 LSB=3000 NAM=PRT LSB=2500 DNOD=[číslo nadřízené stanice] NOD=[číslo řídicího systému] NAM=COM COM=1 IRQ=4 BD=38400 BIT=8 PAR=N STO=2 LRB=2500;

5.4.2.5. DestroyChn

```
procedure DestroyChn; virtual;
```

Po zavolání této metody se zruší řetězec komunikačních objektů odvozených od ChnVirt a uvolní se alokovaná paměť pro přijímací a vysílací buffery.

Pokud před touto funkcí nebyla volána metoda Close, provede se také uvolnění komunikačních objektů odvozených od CoBase.

5.4.2.6. Open

```
function Open : tChnResult; virtual;
```

Po zavolání této metody se zavolají metody ChOpen, ChReceiveBuffer a ChConnect komunikačních objektů odvozených od ChnVirt. Při jejich volání se vytvoří komunikační objekty odvozené od CoBase. Výsledkem volání této metody je uvedení kanálu do stavu, ve kterém lze přijímat a vysílat data.

Po úspěšném volání metody ChConnect se provede nastavení proměnné m_fUseDF2 podle nastavení komunikačního řetězce.

Po zavolání této metody lze měnit pouze některé parametry komunikačních objektů pomocí konfiguračního řetězce a použití metody ChSetParam, viz příslušný manuál komunikační knihovny.

Funkce vrací stav komunikačního kanálu. Pokud nedošlo k chybě vrací se res_Ok.

5.4.2.7. Close

```
function Close : Boolean; virtual;
```

Funkce provede uzavření komunikačního kanálu.

Funkce vrací TRUE, pokud při uzavírání komunikačního kanálu nedošlo k žádné chybě. V opačném případě se vrací FALSE.

5.4.2.8. Register

```
function Register(pObj : PWinTickVirt) : Boolean; virtual;
```

Funkce vrací TRUE, pokud byl funkční objekt úspěšně zařazen do pole m_WinTicks. V opačném případě se vrací FALSE.

Po zařazení funkčního objektu do pole m_WinTicks se provede jeho připojení ke komunikačnímu kanálu. V rámci připojení se především nastaví jeho ukazatele přijímacího a vysílacího bufferu na společnou datovou část komunikačního kanálu pomocí metody Attach. Při jejím volání je důležité mít správně nastavenou položku m_fUseDF2, která rozhoduje o struktuře bufferů – viz protokol DF1 a DF2.

Při zařazování objektu do pole m_WinTicks je zajištěna synchronizace s komunikačním procesem. Proto lze funkční objekty přidávat a ubírat za běhu komunikačního procesu. Dále se během přidávání testuje, zda je typ vkládaného objektu vložen pouze jednou. Pokud typ vkládaného objektu je už zaregistrován, vrací se FALSE.

5.4.2.9. UnRegister

```
function UnRegister(pObj : PWinTickVirt) : Boolean; virtual;
```

Funkce vrací TRUE, pokud byl funkční objekt úspěšně vyřazen z pole WinTicks. V opačném případě se vrací FALSE.

Po vyřazení funkčního objektu z pole m_WinTicks se provede jeho odpojení od komunikačního kanálu. V rámci odpojení se především nastaví jeho ukazatele přijímacího a vysílacího bufferu na NIL.

Při vyřazování objektu z pole m_WinTicks je zajištěna synchronizace s komunikačním procesem. Proto lze funkční objekty přidávat a ubírat za jeho běhu.

5.4.2.10. CreateProcess

```
function CreateProcess:Boolean; virtual;
```

Funkce vrací TRUE, pokud komunikační objekt má nastaveny všechny vyžadované parametry pro vytvoření komunikačního procesu a proces se podaří bez chyby vytvořit. Požadovanými parametry jsou:

- vytvoření komunikačního kanálu – tj. nastavení položek struktury m_ChnDesc a
- spuštění operačního systému ReTOS.

V opačném případě se vrací FALSE.

5.4.2.11. DestroyProcess

```
function DestroyProcess:Boolean; virtual;
```

Funkce vrací TRUE, pokud byl komunikační proces zrušen. V opačném případě se vrací FALSE.

6. Funkční objekty

Další část manuálu bude zaměřena na popis objektů implementujících vybranou funkci vizualizace. Dále se tyto objekty budou označovat funkčními objekty.

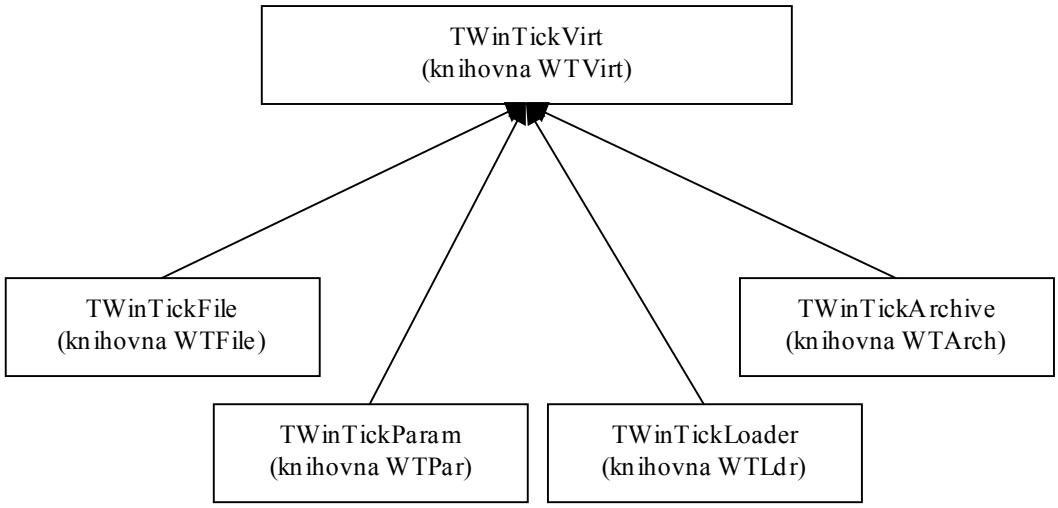
Dále popisované objekty jsou odvozeny od objektu TWinTickVirt, který definuje rozhraní s komunikačním objektem TWinCom. Prostřednictvím tohoto rozhraní se jednotlivým objektům předává řízení a ty v komunikačním automatu provedou zpracování. Protože každá funkce vizualizace používá jiný komunikační automat, liší se uživatelské rozhraní.

Uživatelské rozhraní je vytvořeno pomocí virtuálních metod, jejichž těla jsou v potomcích funkčních objektů předefinována. Ve většině případů se v těle těchto předefinovaných metod doplní pouze funkční kód dané funkce vizualizace (viz přiložené příklady). Metody uživatelského rozhraní jsou od metod rozhraní s komunikačním objektem rozlišeny prefixem **fn**. (Pozn. Na rozdíl od komunikačního rozhraní by funkce uživatelského rozhraní měly být vždy předefinovány.)

Vizualizace poskytuje řídicímu systému následující funkce. V závorkách jsou uvedeny objekty a knihovny implementující požadovanou funkci vizualizace.

- zobrazení archivů (TWinTickArchive, WTArch)
- přenos online parametrů řídicího systému (TWinTickParam, WTPar)
- přenos binárních dat (TWinTickFile, WTFile)
- nahrávání nové verze firmware do řídicího systému (TWinTickLoader, WTLdr)

Graficky lze objektovou stavebnici zobrazit takto:



6.1. Knihovna WTErr

6.1.1. Konstanty

V knihovně jsou definovány chybové kódy pro jednotlivé funkce vizualizace.

Chybové kódy a řetězce

`rc_Ok = 0, str_Ok = ''`

Přijatá zprávy byla zpracována bez chyby.

`rc_NotImplemented = 20, str_NotImplemented = 'Not implemented'`

Chybový kód se vrací, pokud uživatel zapomněl předefinovat nějakou metodu objektu nebo vizualizace vyžaduje předání nepodporovaného parametru.

`rc_ErrCmd = 254, str_ErrCmd = 'Unknown command'`

Chybový kód se vrací, pokud komunikační objekt nenašel obsluhu pro přijatou zprávu. Zapomenuté volání funkce Register.

`rc_Err = 255, str_Err = 'Unspecified fault'`

Chybový kód znamená nespecifikovanou chybu.

6.2. Knihovna WTVirt

Knihovna obsahuje implementaci předka funkčních objektů vizualizace TWinTickVirt. Tento předek definuje rozhraní s komunikačním objektem TWinCom.

6.2.1. Konstanty

```
cVer  = 'v2.02, 5.2.2003'
```

konstanta udává celkovou verzi a poslední změnu knihoven KitKing, tzn. všech jednotek tvořící programový balík knihoven KitKing.

Identifikační kódy funkčních objektů

Funkční kódy se používají při kontrole zaregistrovaných funkčních objektů. Tato kontrola se provádí při volání metody Register u objektu TWinCom.

```
wt_Virt = 1
```

Identifikační kód pro virtuálního předka funkčních objektů TWinTickVirt.

```
wt_File = 2
```

Identifikační kód pro funkční objekt odvozený od objektu TWinTickFile.

```
wt_Archive = 3
```

Identifikační kód pro funkční objekt odvozený od objektu TWinTickArchive.

```
wt_Loader = 4
```

Identifikační kód pro funkční objekt odvozený od objektu TWinTickLoader.

```
wt_Param = 5
```

Identifikační kód pro funkční objekt odvozený od objektu TWinTickParam.

Dále popisovaná konstanta je definována v jednotce XTKing_1.pas. V této jednotce jsou definovány konstanty a typy, které se sdílí mezi vizualizací a řídicím systémem.

Popis konstanty je zde uveden pro úplnost uživatelského rozhraní.

```
g_ResultSzMaxIndx  = 64
```

konstanta se používá při deklaraci typu Tgcmd_Result.

6.2.2. Typy

Dále popisovaný typ je definován v jednotce XTKing_1.pas. V této jednotce jsou definovány konstanty a typy, které se sdílí mezi vizualizací a řídicím systémem.

Popis typu je zde uveden pro úplnost uživatelského rozhraní.

```
Tgcmd_Result = record
 ResultCode : word;          { 0=OK, ostatni kod chyby }
 ResultSz   : array[0..g_ResultSzMaxIndx] of char;
                             { null terminated str }
end;
```

Typ se používá pro přenos návratového kódu zpravidla chybového některých operací, viz metoda SetResultCode u potomků objektu TWinTickVirt.

Dále popisovaný typ je definován v jednotce ChnSofT.pas. V této jednotce jsou definovány konstanty a typy, které používá komunikační knihovna ChnSofs2.

Popis typu je zde uveden pro úplnost uživatelského rozhraní.

```
pMessDF1 = ^tMessDF1;
tMessDF1 = record
 Cmd        : Byte;          { identifikace zpravy }
 Rec        : tMess;         { vlastni prenasena data }
end;
```

6.2.3. Funkce

```
function ExLockKernel:Boolean;
```

Protože během vizualizace je potřeba zajišťovat synchronizace, je v této knihovně obsažena funkce ExLockKernel. Tato funkce řeší hazardní stav při běžném používání funkce LockKernel. Funkce vrací TRUE, pokud jádro operačního systému už bylo uzamčeno. V případě, že jádro ještě nebylo uzamčeno, provede se jeho uzamknutí a funkce vrátí FALSE.

Tuto funkci nelze použít při synchronizaci práce s proměnnými, do kterých se zapisuje v přerušení. V případě těchto proměnných se musí provádět synchronizace pomocí příkazů assembleru PUSHF, CLI a POPF.

6.2.4. Objekt TWinTickVirt

Objekt implementuje rozhraní s komunikačním objektem TWinCom a slouží jako předek funkčních objektů vizualizace. Protože potomci tohoto objektu můžou být v komunikačním objektu zaregistrovány pouze jednou, měly by se používat jako správci, tj. všech archívů v aplikaci apod.

6.2.4.1. Položky

```
m_pRMess   : pMessDF1;
```

Ukazatel na buffer s přijatou zprávou.

```
m_pSMess   : pMessDF1;
```

Ukazatel na buffer zprávy určené k vysílání.

6.2.4.2. Obecné metody

6.2.4.2.1. Konstruktor Init

```
constructor Init;
```

Konstruktor slouží k dokončení inicializace objektu s virtuálními metodami a jeho položek. Položky objektu jsou inicializovány následovně.

Položky m_pRMess a m_pSMess jsou nastaveny na NIL.

6.2.4.2.2. Destruktor Done

```
destructor Done; virtual;
```

Destruktor slouží ke zrušení objektu.

6.2.4.3. Metody rozhraní s komunikačním objektem

6.2.4.3.1. Attach

```
procedure Attach(pRMess:pMessDF1; pSMess:pMessDF1); virtual;
```

Metoda, která se musí volat před zařazením funkčního objektu do smyčky zpracovávající přijaté zprávy a provádí inicializaci ukazatelů na datovou část komunikačních bufferů s ohledem na používaný protokol DF1 a DF2.

Objekt TWinTickVirt nastaví položky m_pRMess a m_pSMess na předané parametry.

6.2.4.3.2. Detach

```
procedure Detach; virtual;
```

Metoda, která se automaticky volá po volání funkce UnRegister objektu TWinCom. V těle této metody se provádí deinicializace komunikačních bufferů.

Objekt TWinTickVirt nastaví položky m_pRMess a m_pSMess na NIL.

6.2.4.3.3. GetId

```
function GetId:Byte; virtual;
```

Funkce vrací identifikaci objektu.

Objekt TWinTickVirt vrací wt_Virt.

6.2.4.3.4. Tick

```
function Tick:Boolean; virtual;
```

Funkce provádí zpracování přijaté zprávy a připravení odpovědi na tuto zprávu. Funkce vrací TRUE pokud přijatá zpráva byla obsloužena. V opačném případě vrací FALSE. Podrobnější popis zpracování přijatých zpráv najdete v kapitole 5.1.

V těle této funkce jsou implementovány komunikační automaty, při jejichž zpracování se volají virtuální metody funkčních objektů. Protože každá funkce vizualizace používá jiný komunikační automat, je u každého objektu různé uživatelské rozhraní.

Objekt TWinTickVirt vždy vrací FALSE.

6.2.4.3.5. SetResultCode

```
procedure SetResultCode(resCode:Word); virtual;
```

Metoda nastaví návratový kód dle komunikačního automatu.

Objekt TWinTickVirt nastaví návratový kód používaný při neimplementované funkci vizualizace (rc_NotImplemented) a při bezchybném zpracovaní přijaté zprávy (rc_Ok). Pro odpověď se používají zprávy typu gcmd_Result.

Při přetížení této metody se musí nastavit typ zprávy na gcmd_Result v položce m_pSMess^.Cmd, předaný návratový kód resCode do položky Tgcmd_Result(Addr(m_pSMess^.Rec)^).ResultCode a řetězec popisující návratový kód do položky Tgcmd_Result(Addr(m_pSMess^.Rec)^). ResultSz. Protože nastavovaný řetězec není typu STRING, ale PCHAR měla by se při jeho nastavování použít funkce StrPCopy. Max. velikost řetězce včetně ukončovacího znaku #0 nesmí v případě použití typu Tgcmd_Result přesáhnout 64B, v ostatních případech je možné přenášet delší řetězce. Přijaté řetězce jsou ve vizualizaci zobrazeny v chybovém dialogu.

Někteří potomci tuto funkci používají pouze při vracení chybových stavů a jinak používají vlastní typy zpráv. Tito potomci mají různé typy zpráv pro chybu při zpracování přijaté zprávy a pro potvrzení zpracování přijaté zprávy.

6.2.4.3.6. GetMessSize

```
function GetMessSize:Word; virtual;
```

Funkce vrací délku zprávy připravené k odvysílání. (Pozn. Na rozdíl od předchozích verzí (do verze 1.02) tato délka neobsahuje velikost hlavičky protokolu DF2.)

Objekt TWinTickVirt vysílá pouze zprávy typu gcmd_Result, proto funkce implementuje výpočet délky pouze u těchto zpráv. Pro ostatní typy zpráv se vrací 0.

Pozn.: Při výpočtu délky zprávy typu gcmd_Result se používá funkce StrLen, která vrací délku řetězce typu PCHAR (tj. poslední znak v řetězci končí 0).

K této délce řetězce jsou přičteny hlavičky, poslední znak řetězce (tj. 0) a návratový kód.

6.3. Knihovna WTPar

Knihovna obsahuje implementaci funkčního objektu TWinTickParam, který umožňuje přenášet online parametry (proměnné aplikace) řídicího systému – jednotlivě nebo blokově. Přenášené parametry lze použít i jako povely řídicího systému.

Deklarace uživatelských logických adres, dále LA, online parametrů musí začínat s offsetem 100 (konstanta BaseLA_Firmware), protože logické adresy od 0..99 jsou používány pro předdefinované parametry. Tyto parametry jsou deklarovány v souboru XTKing_1.pas a jsou například používány pro práci s restart strukturou a bitovými proměnnými.

6.3.1. Typy

Popisovaný typ je definován v jednotce XTKing_1.pas. V této jednotce jsou definovány konstanty a typy, které se sdílí mezi vizualizací a řídicím systémem.

Popis typu je zde uveden pro úplnost uživatelského rozhraní.

```
Tgcmd_ParamValueBlk = record
 LACnt      : Word;          { pocet LA v nasledujicich poli LAArr
                             a DataArr }
 LAArr      : array [0..LACnt-1] of Word;
                             { pole pozadovanych online parametru
                             ridiciho systemu }
 DataArr    : array [0..LACnt-1] of Byte;
                             { datovy buffer pro ulozeni pozadovanych
                             hodnot ridiciho systemu }
end;
```

Typ sloužící pro čtení bloku online parametrů z řídicího systému.

```
TWrByteBits = record
 DestByteLA : word;          { logicka adresa online parametru o
                             velikosti byte, na kterem se transakce
                             provede }
 ByteSetMsk :byte;           { maska pro bity, u nichz se pozaduje
                             nastaveni do '1' }
 ByteResMsk :byte;           { maska pro bity, u nichz se pozaduje
                             nastaveni do '0' }
end;
```

Následující typy slouží pro vykonávání bitové operace u online parametru s velikostí BYTE.

```
TWrWordBits = record
 DestWordLA : word;          { logicka adresa online parametru o
                             velikosti word, na kterem se transakce
                             provede }
 WordSetMsk :byte;           { maska pro bity, u nichz se pozaduje
                             nastaveni do '1' }
 WordResMsk :byte;           { maska pro bity, u nichz se pozaduje
                             nastaveni do '0' }
end;
```

Následující typy slouží pro vykonávání bitové operace u online parametru s velikostí WORD.

```
TWrDWordBits = record
 DestDWordLA  : word;       { logicka adresa online parametru o
                              velikosti word, na kterem se transakce
                              provede }
 DWordSetMsk  :byte;        { maska pro bity, u nichz se pozaduje
                              nastaveni do '1' }
 DWordResMsk  :byte;        { maska pro bity, u nichz se pozaduje
                              nastaveni do '0' }
end;
```

Následující typy slouží pro vykonávání bitové operace u online parametru s velikostí DWORD.

6.3.2. Objekt TWinTickParam

Objekt implementuje přenos online parametrů řídicího systému.

6.3.2.1. Obecné metody

6.3.2.1.1. Konstruktor Init

```
constructor Init;
```

Konstruktor slouží k dokončení inicializace objektu s virtuálními metodami a jeho položek. V těle metody se volá konstruktor předka.

6.3.2.1.2. Destruktor Done

```
destructor Done; virtual;
```

Destruktor slouží ke zrušení objektu. V těle metody se volá destruktor předka.

6.3.2.2. Metody rozhraní s komunikačním objektem

6.3.2.2.1. GetId

```
function GetId:Byte; virtual;
```

Funkce vrací identifikaci objektu. Objekt vrací wt_Param.

6.3.2.2.2. Tick

```
function Tick:Boolean; virtual;
```

Funkce provádí zpracování přijaté zprávy a připravení odpovědi na tuto zprávu. Funkce vrací TRUE, pokud přijatá zpráva byla obsloužena. V opačném případě se vrací FALSE. Podrobnější popis zpracování přijatých zpráv najdete v kapitole 5.1.

V těle této funkce je implementován komunikační automat, který provádí přenos online parametrů z nadřízeného systému do řídicího systému a opačně. V jednotlivých stavech automatu se volají funkce s prefixem fn, dále funkce uživatelského rozhraní objektu. Tyto funkce musí být v potomku objektu předefinovány. Ve většině případů se doplní pouze funkční tělo funkce. (viz dodávané příklady)

6.3.2.3. GetMessSize

```
function GetMessSize:Word; virtual;
```

Funkce vrací délku zprávy připravené k odvysílání. (Pozn. Na rozdíl od předchozích verzí (do verze 1.02) tato délka neobsahuje velikost hlavičky protokolu DF2.)

Objekt implementuje výpočet délky zprávy typu gcmd_PutParamVal, gcmd_PutParamValBlk a pro ostatní typy zpráv volá GetMessSize předka.

6.3.2.4. Metody uživatelského rozhraní objektu

Následující metody musí být předefinovány a v jejich popisu bude popsán funkční kód, který se do nich musí doplnit.
LA se používá ve významu logická adresa.

6.3.2.4.1. fnGetParamVal

```
function fnGetParamVal(LA:Word; var Data):Word; virtual;
```

Funkce přečte online parametr řídicího systému do datového bufferu Data podle požadované adresy LA. Funkce vrací rc_Ok, pokud buffer Data byl nastaven. V opačném případě funkce vrací rc_NotImplemented. Pokud by se měl vracet jiný návratový kód než rc_NotImplemented, musí být implementována funkce SetResultCode.
Uživatel musí zajistit synchronizaci při čtení hodnoty online parametru.
Pozn. Při přiřazování hodnoty do bufferu Data nedoporučujeme používat příkaz Move, protože obchází veškeré typové kontroly, což by při pozdější změně typu mohlo vést k těžko odhalitelné chybě.

6.3.2.4.2. fnPutParamVal

```
function fnPutParamVal(LA:Word; var Data):Word; virtual;
```

Funkce nastaví online parametr řídicího systému na adrese LA na hodnotu předanou v datovém bufferu Data. Funkce vrací rc_Ok, pokud byl parametr nastaven. V opačném případě funkce vrací rc_NotImplemented. Pokud by se měl vracet jiný návratový kód než rc_NotImplemented, musí být implementována funkce SetResultCode.
Uživatel musí zajistit synchronizaci při nastavování hodnoty do online parametru.
Zápis hodnoty do online parametru může být také interpretován jako požadavek na vykonání určitého příkazu.
Pozn. Při přiřazování hodnoty do proměnné aplikace nedoporučujeme používat příkaz Move, protože obchází veškeré typové kontroly, což by při pozdější změně typu mohlo vést k těžko odhalitelné chybě.

6.3.2.4.3. fnGetParamValBlk

```
function fnGetParamValBlk(var ParamValBlk:Tgcmd_ParamValueBlk):
        Word; virtual;
```

Funkce by měla být předefinována pouze v případě, kdyby uživatel chtěl zrychlit přístupy k jednotlivým parametrům – např. kopírování připravené struktury do datového bufferu nebo zmenšit dobu zamknutí jádra. Při předefinování funkce musí být zajištěna synchronizace při přístupu k požadovaným parametrům.
Funkce zamkne jádro a potom voláním funkce fnGetParamVal případně fnGetFirmwareParamVal čte jednotlivé parametry řídicího systému. Parametry jsou zadány polem adres LA a jejich hodnoty se zapisují do datového bufferu DataArr. Po přečtení všech parametrů se jádro odemkne.
Funkce vrací rc_Ok, pokud byl datový buffer nastaven. V opačném případě funkce vrací rc_NotImplemented. Pokud by se měl vracet jiný návratový kód než rc_NotImplemented, musí být implementována funkce SetResultCode.
Uživatel musí zajistit synchronizaci při čtení hodnot online parametrů.
Pozn. Při přiřazování hodnoty do bufferu ParamValBlk nedoporučujeme používat příkaz Move, protože obchází veškeré typové kontroly, což by při pozdější změně typu mohlo vést k těžko odhalitelné chybě.

6.3.2.4.4. fnPutParamValBlk

```
function fnPutParamValBlk(var ParamValBlk:Tgcmd_ParamValueBlk):
        Word; virtual;
```

Funkce by měla být předefinována pouze v případě, kdyby uživatel chtěl zrychlit přístupy k jednotlivým parametrům – např. kopírování datového bufferu do připravené struktury nebo zmenšit dobu zamknutí jádra. Při předefinování funkce musí být zajištěna synchronizace při přístupu k požadovaným parametrům.

Funkce zamkne jádro a potom voláním funkce fnPutParamVal případně fnPutFirmwareParamVal nastavuje jednotlivé parametry řídicího systému. Parametry, které se nastavují jsou zadány polem adres LA a jejich hodnoty jsou uloženy v datovém bufferu DataArr. Po nastavení všech parametrů se jádro odemkne.

Funkce vrací rc_Ok, pokud byl datový buffer nastaven. V opačném případě funkce vrací rc_NotImplemented. Pokud by se měl vracet jiný návratový kód než rc_NotImplemented, musí být implementována funkce SetResultCode.

Uživatel musí zajistit synchronizaci při nastavování hodnot online parametrů.

Pozn. Při přiřazování hodnoty do proměnných aplikace nedoporučujeme používat příkaz Move, protože obchází veškeré typové kontroly, což by při pozdější změně typu mohlo vést k těžko odhalitelné chybě.

6.3.2.4.5. fnGetParamSize

```
function fnGetParamSize(LA:Word):Word; virtual;
```

Funkce vrací velikost parametru řídicího systému podle adresy LA. Při implementaci této funkce je vhodné při malém počtu online parametrů použít CASE. Při větším počtu online parametrů je lepší použít tabulku, která se definuje v souboru popisujícím jednotlivé parametry řídicího systému. (např. PutParSizeTab v souboru xCrExam1.pas)

6.3.2.4.6. fnGetParamBlkSize

```
function  fnGetParamBlkSize(var ParamValBlk:Tgcmd_ParamValueBlk):
        Word; virtual;
```

Funkce vrací velikost bloku přenášených dat ve struktuře ParamValBlk.

Funkce by měla být předefinována pouze v případě kdyby uživatel chtěl zrychlit výpočet.

Při výpočtu velikosti bloku přenášených dat se v této funkci volá funkce fnGetParamSize pro jednotlivé parametry a součet jejich velikostí je výslednou velikostí bloku dat. Parametry jsou zadány polem adres LA.

6.3.2.4.7. fnGetFirmwareParamVal

```
function  fnGetFirmwareParamVal(LA:Word; var Data):Word; virtual;
```

Funkce slouží pro čtení online parametrů na úrovni firmware při zákaznických úpravách programu TheKing nebo složitějších operací nad datovými typy.

Funkce vrací rc_Ok pokud požadovaná operace byla provedena. V opačném případě vrací rc_NotImplemented. Pokud by se měl vracet jiný návratový kód než rc_NotImplemented, musí být implementována funkce SetResultCode.

Předek neimplementuje žádnou obsluhu online parametrů:

V aplikaci je doporučováno provést obsluhu následujícího online parametru:

LA_RstStruct — do proměnné Data se nastaví struktura typu TRstStruct. Obsluha tohoto parametru je ukázána v dodávaném příkladu Example1 a lze jí ve většině případů zkopírovat.

Pozn. Při přiřazování hodnoty do bufferu Data nedoporučujeme používat příkaz Move, protože obchází veškeré typové kontroly, což by při pozdější změně typu mohlo vést k těžko odhalitelné chybě.

6.3.2.4.8. fnPutFirmwareParamVal

```
function  fnGetFirmwareParamVal(LA:Word; var Data):Word; virtual;
```

Funkce slouží pro nastavení online parametrů na úrovni firmware při zákaznických úpravách programu TheKing nebo složitějších operací nad datovými typy.

Funkce vrací rc_Ok pokud požadovaná operace byla provedena. V opačném případě vrací rc_NotImplemented. Pokud by se měl vracet jiný návratový kód než rc_NotImplemented, musí být implementována funkce SetResultCode.

Předek implementuje obsluhu těchto online parametrů:

LA_WrByteBits v proměnné Data je předána struktura typu TWrByteBits. Obsluha této bitové operace probíhá volání funkce fnGetParamVal a přečtení hodnoty parametru z logické adresy DestByteLA do lokálního bufferu, poté se provedou bitové operace

buffer:=(buffer and (not ByteResMsk)) or ByteSetMsk.

Po vypočtení nové hodnoty parametru se tato hodnota uloží do parametru na logické adrese DestByteLA voláním funkce fnPutParamVal.

LA_WrWordBits v proměnné Data je předána struktura typu TWrWordBits. Obsluha této bitové operace probíhá volání funkce fnGetParamVal a přečtení hodnoty parametru z logické adresy DestWordLA do lokálního bufferu, poté se provedou bitové operace

buffer:=(buffer and (not WordResMsk)) or WordSetMsk.

Po vypočtení nové hodnoty parametru se tato hodnota uloží do parametru na logické adrese DestWordLA voláním funkce fnPutParamVal.

LA_WrDWordBits v proměnné Data je předána struktura typu TWrDWordBits.

Obsluha této bitové operace probíhá volání funkce fnGetParamVal a přečtení hodnoty parametru z logické adresy DestDWordLA do lokálního bufferu, poté se provedou bitové operace

buffer:=(buffer and (not DWordResMsk)) or DWordSetMsk.

Po vypočtení nové hodnoty parametru se tato hodnota uloží do parametru na logické adrese DestDWordLA voláním funkce fnPutParamVal.

V aplikaci je doporučováno provést obsluhu následujícího online parametru:

LA_ClrStruct v proměnné Data je předána struktura typu TRstStruct. Obsluha tohoto parametru je ukázána v dodávaném příkladu Example1 a lze jí ve většině případů zkopírovat.

Pozn. Při přiřazování hodnoty do proměnné aplikace nedoporučujeme používat příkaz Move, protože obchází veškeré typové kontroly, což by při pozdější změně typu mohlo vést k těžko odhalitelné chybě.

6.4. Knihovna WTLdr

Knihovna obsahuje implementaci funkčního objektu TWinTickLoader, který umožňuje zápis nové verze firmware do řídicího systému.

6.4.1. Typy

Dále popisovaný typ je definován v jednotce XTKing_1.pas. V této jednotce jsou definovány konstanty a typy, které se sdílí mezi vizualizací a řídicím systémem.

Popis typu je zde uveden pro úplnost uživatelského rozhraní.

```
TLoaderAppInfo = record
 AppTypeID  : Byte;          { typ komunikující aplikace
                             0 - neznama aplikace
                             1 - LOADER
                             2 - ridici system - firmware }
 AppInfoStr : TLoaderAppInfoStr; (String[40])
                             { identifikacni retezec aplikace }
end;
```

Struktura slouží pro zjištění typu aplikace a identifikačního řetězce.

6.4.2. Objekt TWinTickLoader

Objekt implementuje pomocné funkce pro práci s programem LOADER, který umožňuje zápis firmware do řídicího systému pomocí vizualizace.

6.4.2.1. Obecné metody

6.4.2.1.1. Konstruktor Init

```
constructor Init;
```

Konstruktor slouží k dokončení inicializace objektu s virtuálními metodami a jeho položek. V těle metody se volá konstruktor předka.

6.4.2.1.2. Destruktor Done

```
destructor Done; virtual;
```

Destruktor slouží ke zrušení objektu. V těle metody se volá destruktor předka.

6.4.2.2. Metody rozhraní s komunikačním objektem

6.4.2.2.1. GetId

```
function GetId:Byte; virtual;
```

Funkce vrací identifikaci objektu. Objekt vrací wt_Loader.

6.4.2.2.2. Tick

```
function Tick:Boolean; virtual;
```

Funkce provádí zpracování přijaté zprávy a připravení odpovědi na tuto zprávu. Funkce vrací TRUE, pokud přijatá zpráva byla obsloužena. V opačném případě vrací FALSE. Podrobnější popis zpracování přijatých zpráv najdete v kapitole 5.1.

V těle této funkce je implementován komunikační automat s obsluhou příkazů potřebných pro zápis nové verze firmware. V jednotlivých stavech automatu se volají funkce s prefixem fn, dále funkce uživatelského rozhraní objektu. Tyto funkce musí být v potomku objektu předefinovány. Ve většině případů se doplní pouze funkční tělo funkce. (viz dodávané příklady)

6.4.2.3. GetMessSize

```
function GetMessSize:Word; virtual;
```

Funkce vrací délku zprávy připravené k odvysílání. (Pozn. Na rozdíl od předchozích verzí (do verze 1.02) tato délka neobsahuje velikost hlavičky protokolu DF2.)

Objekt implementuje výpočet délky zprávy typu wcmd_Loader_QueryAck a pro ostatní typy zpráv volá GetMessSize předka.

6.4.2.4. Metody uživatelského rozhraní objektu

Následující metody musí být předefinovány a v jejich popisu bude popsán funkční kód, který se do nich musí doplnit.

6.4.2.4.1. fnStartLoader

```
procedure fnStartLoader; virtual;
```

Funkce by měla zajistit bezhazardové ukončení aplikace a spuštění aplikace LOADER. Spuštění LOADER je možné provést buď nastavením proměnné gpLdrDesc^.bParForLdr (Pozn. po nastavení této proměnné se musí přepočítat CRC této struktury) nebo poškozením aplikace voláním funkce AtmDestruct nebo AtmTotalDestruct. (Pozn. V aplikaci se musí poškodit modul obsahující jednotku **--LOADER--**). Po ukončení této funkce se v komunikačním objektu provede RESET řídicího systému.

6.4.2.4.2. fnResetApp

```
procedure fnResetApp; virtual;
```

Funkce by měla zajistit bezhazardové ukončení aplikace. Po ukončení této funkce se v komunikačním objektu provede RESET řídicího systému.

6.4.2.4.3. fnQueryApp

```
procedure fnQueryApp(var LdrAppInfo:TLoaderAppInfo); virtual;
```

Funkce nastavuje identifikační řetězec a typ aplikace. Tento řetězec se zobrazuje v okénku vizualizace a slouží pro rozlišení firmware řídicího systému nebo aplikace LOADER. Standardně je typ aplikace ve struktuře LdrAppInfo nastaven na firmware řídicího systému, tj. cLoaderAppInfo_UserApp.

6.5. Knihovna WTFile

Knihovna obsahuje implementaci funkčního objektu TWinTickFile, který umožňuje přenos binárních dat. Dále budou binární data označována jako banky souborů, protože čtení a nastavování hodnot probíhá pomocí souborů. Banka souboru (binární data) je především používána pro nastavování různých konstant, které jsou ukládány do souborů případně jsou z těchto souborů nastavovány.

6.5.1. Konstanty

Chybové kódy

Popisované konstanty jsou definovány v jednotce XTKing_1.pas. V této jednotce jsou definovány konstanty a typy, které se sdílí mezi vizualizací a řídicím systémem.

Popis konstant je uveden pro úplnost uživatelského rozhraní.

```
erc_file_OK                 = 0;
```

Obsluha přijaté zprávy proběhla bez chyby.

```
erc_file_NoImplemented      = 1;
```

Obsluha není implementována.

```
erc_file_CannotBeExecuted   = 2;
```

Obsluha nemůže být dokončena, např. neplatná konverze banku souboru.

```
erc_file_NotAllowed         = 3;
```

Obsluha nelze v daném stavu provést.

```
erc_file_InsufficientMemory= 4;
```

Vybraný soubor je větší než kapacita požadované banky souborů.

```
erc_file_BlockNotWritten    = 5;
```

Požadovaný blok dat nebyl zapsán do banky souborů, např. nelze zapsat do paměti FLASH.

6.5.2. Typy

Popisované typy jsou definovány v jednotce XTKing_1.pas. V této jednotce jsou definovány konstanty a typy, které se sdílí mezi vizualizací a řídicím systémem.

Popis typů je uveden pro úplnost uživatelského rozhraní.

```
TFileBankListInfo = record
 FileBankListCount          : Word;
                            { počet bank souborů v řídicím systému }
end;
```

Struktura slouží pro zjištění počtu bank souborů v řídicím systému.

```
TFileDataHeader = record
 NextHeaderFl                : Boolean;    { následuje další hlavička
                             }
 FileBankNum                 : Byte; { označení banky souboru - nemusí
                             odpovídat adrese banky souboru }
 FileBankName                : TFileBankName; { String[16] }
                             { označení banky }
 FileName                    : TDataFileName; { String[80] }
                             { jméno souboru zavedeného do banky }
 FileSize                    : LongInt;    { velikost souboru
                             uloženého v  bance }
 FileDaTi                    : LongInt;    { čas souboru uloženého v
                             bance - používá se DOS formát }
 FileAttr                    : LongInt;    { atributy souboru
                             uloženého v bance }
end;
TFileBankDataInfo = TFileDataHeader;
TPutFileBankDataOpen = TFileDataHeader;
```

Struktura slouží pro zjištění informací o souboru, z kterého se banka inicializovala případně, do kterého se mají data uložit.

```
TDataBlockFile = record
 NextBlockFl                 : Boolean;    { následuje další blok
                             binárních dat }
 DataFileOffs                : LongInt;    { offset čtených nebo
                             nastavovaných dat v bance souboru }
 BlockSize                   : Word  { velikost bloku čtených nebo
                             nastavovaných dat v bance souboru }
 Block                       : TBinBlockFile; { array [0..511] of
                             Byte }
                             { hodnoty čtených nebo nastavovaných dat
                             v bance souboru }
end;
```

Struktura slouží pro čtení nebo nastavení dat v bance souboru.

Dále popisovaný typ je definován v jednotce ChnSofT.pas. V této jednotce jsou definovány konstanty a typy, které používá komunikační knihovna ChnSofs2.

Popis typu je zde uveden pro úplnost uživatelského rozhraní.

```
pMessDF1 = ^tMessDF1;
tMessDF1 = record
 Cmd         : Byte;          { identifikace zpravy }
 Rec         : tMess;         { vlastni prenasena data }
end;
```

6.5.3. Objekt TWinTickFile

Objekt implementuje funkce pro přenos binárních dat.

6.5.3.1. Položky

`m_pAck : pFILE_ACK;`

Ukazatel na buffer vysílané zprávy potvrzující úspěšné provedení přijaté zprávy.

`m_pErr : pFILE_ERR;`

Ukazatel na buffer vysílané zprávy signalizující chybu při zpracování přijaté zprávy.

`m_pCmd : pFILE_CMD;`

Ukazatel na buffer s přijatou zprávou, který je přetypován na strukturu obsahující požadovanou operaci s bankou souboru a daty potřebnými k jejímu dokončení.

`m_bFile : Byte;`

Index banky souboru, ke kterému se právě přistupuje.

```
m_liFileOffs                : LongInt;
```

Offset do banky souboru, ze které se data právě čtou nebo do které se zapisují.

6.5.3.2. Obecné metody

6.5.3.2.1. Konstruktor Init

```
constructor Init;
```

Konstruktor slouží k dokončení inicializace objektu s virtuálními metodami a jeho položek. V těle metody se volá konstruktor předka a potom se nastaví položky m_pAck, m_pErr a m_pCmd na NIL. Položky m_bFile a m_liFileOffs se nastaví na 0.

6.5.3.2.2. Destruktor Done

```
destructor Done; virtual;
```

Destruktor slouží ke zrušení objektu. V těle metody se volá destruktor předka.

6.5.3.3. Metody rozhraní s komunikačním objektem

6.5.3.3.1. Attach

```
procedure Attach(pRMess:pMessDF1; pSMess:pMessDF1); virtual;
```

Metoda, která se musí volat před zařazením funkčního objektu do smyčky zpracovávající přijaté zprávy. V těle metody se zavolá Attach předka, pak se nastaví položky m_pAck, m_pErr na datovou část vysílacího bufferu a m_pCmd na datovou část přijímacího bufferu komunikačního kanálu. Nakonec jsou položky m_bFile a m_liFileOffs nastaveny na 0.

6.5.3.3.2. Detach

```
procedure Detach; virtual;
```

Metoda, která se automaticky volá po volání funkce UnRegister objektu TWinCom. V těle této metody se provádí deinicializace komunikačních bufferů.

V těle metody se nastaví m_pAck, m_pErr a m_pCmd na NIL a pak se zavolá Detach předka.

6.5.3.3.3. GetId

```
function GetId:Byte; virtual;
```

Funkce vrací identifikaci objektu. Objekt vrací wt_File.

6.5.3.3.4. Tick

```
function Tick:Boolean; virtual;
```

Funkce provádí zpracování přijaté zprávy a připravení odpovědi na tuto zprávu. Funkce vrací TRUE, pokud přijatá zpráva byla obsloužena. V opačném případě vrací FALSE. Podrobnější popis zpracování přijatých zpráv najdete v kapitole 5.1.

V těle této funkce je implementován komunikační automat s obsluhou příkazů potřebných pro přenos binárních dat. V jednotlivých stavech automatu se volají funkce s prefixem fn, dále funkce uživatelského rozhraní objektu. Tyto funkce musí být v potomku objektu předefinovány. Ve většině případů se doplní pouze funkční tělo funkce. (viz dodávané příklady)

6.5.3.3.5. SetResultCode

```
procedure SetResultCode(resCode:Word); virtual;
```

Metoda nastaví návratový kód odpovídající obsluze přenosu binárních dat. Pro odpověď se používají zprávy typu wcmd_FILE_ERR.

6.5.3.4. GetMessSize

```
function GetMessSize:Word; virtual;
```

Funkce vrací délku zprávy připravené k odvysílání. (Pozn. Na rozdíl od předchozích verzí (do verze 1.02) tato délka neobsahuje velikost hlavičky protokolu DF2.)

Objekt implementuje výpočet délky zprávy typu wcmd_FILE_ACK, wcmd_FILE_ERR a pro ostatní typy zpráv volá GetMessSize předka.

6.5.3.5. Metody uživatelského rozhraní objektu

Následující metody musí být předefinovány a v jejich popisu bude popsán funkční kód, který se do nich musí doplnit.

6.5.3.5.1. fnConvertFileNo

```
function fnConvertFileNo(bFileNo:Byte):Byte; virtual;
```

Funkce vrací index banky souboru podle předaného označení banky souboru. Index souboru musí být vždy mezi [0..n-1].

V případě, že označení banku souboru je stejný jako index banku nemusí se tato funkce předefinovat, protože vrací bFileNo.

Počet všech bank souborů lze získat pomocí funkce fnGetFileCnt.

6.5.3.5.2. fnGetFileCnt

```
function fnGetFileCnt:Byte; virtual;
```

Funkce vrací počet všech bank souborů v řídicím systému.

6.5.3.5.3. fnAbort

```
procedure fnAbort; virtual;
```

Metoda zajistí bezhazardové ukončení práce se soubory bez ohledu na současný stav komunikace. Povel se posílá jako reakce na uživatelský vstup – stisk tlačítka Abort.

Po volání této metody se nastaví položka m_bFileNo na 0.

6.5.3.5.4. fnClose

```
function fnClose:Byte; virtual;
```

Funkce zajistí bezhazardové ukončení práce se soubory. Při ukončování práce se soubory se provede kontrola současného stavu komunikačního automatu a pokud nedošlo k chybě vrací se erc_file_Ok. Pokud došlo k chybě, vrací se návratový kód různý od erc_file_Ok.

Po volání této funkce se nastaví položky m_bFileNo a m_liFileOffs na 0.

6.5.3.5.5. fnGetBankListOpen

```
procedure fnGetBankListOpen
        (var FileBankListInfo:TFileBankListInfo); virtual;
```

Metoda vrací v parametru FileBankListInfo.FileBankListCount počet všech bank souborů v řídicím systému. Tato hodnota se získá pomocí fnGetFileCnt, proto není potřeba tuto funkci předefinovat.

Po volání této metody se nastaví položka m_bFileNo na 0.

6.5.3.5.6. fnGetBankListItem

```
procedure fnGetBankListItem
        (var FileBankListItem:TFileDataHeader); virtual;
```

Metoda vrací v parametru FileBankListItem informace o souboru, který je spojen s aktivní bankou souborů. Informace o aktivní bance souborů je uložena v položce m_bFileNo, která se automaticky inkrementuje v komunikačním automatu po volání této funkce.

Před voláním této funkce se kontroluje položka m_bFileNo na max. počet všech bank souborů. Pokud je tento počet překročen vrací se chybový kód – erc_file_CannotBeExecuted.

6.5.3.5.7. fnGetBankDataOpen

```
function fnGetBankDataOpen
        (var FileBankDataInfo:TFileBankDataInfo):Byte; virtual;
```

Funkce otevře požadovanou banku souborů pro přenos dat z řídicího systému do nadřízeného počítače s aplikací vizualizace. Hodnota požadované banky je uložena v položce m_bFileNo.

Před voláním této funkce se provede konverze označení banky souboru pomocí volání fnConvertFileNo a nastaví se položka m_bFile. Poté se provede kontrola této položky na max. počet všech bank souborů. Pokud je tento počet překročen vrací se chybový kód – erc_file_CannotBeExecuted.

Pokud banku souborů nelze otevřít, vrací se chybová hodnota, která je různá od erc_file_Ok. V opačném případě se vrací erc_file_Ok.

Po volání této funkce se nastaví položka m_liFileOffs na 0.

6.5.3.5.8. fnGetBankDataBlock

```
function fnGetBankDataBlock
        (var FileBankDataBlock:TDataBlockFile):Byte; virtual;
```

Metoda provede nastavení datového bufferu FileBankDataBlock.Block a dalších položek v proměnné FileBankDataBlock. V položce m_bFileNo je nastavena hodnota indexu aktuální banky souborů. Položka m_liFileOffs se může použít jako pomocná proměnná, ve které je uložena poslední pozice přečtených binárních dat.

Pokud lze blok dat z aktuální banky souborů bez chyby přečíst, vrací se erc_file_Ok. V opačném případě se vrací hodnota různá od erc_file_Ok.

6.5.3.5.9. fnPutBankDataOpen

```
function fnPutBankDataOpen
        (var FileBankDataOpen:TPutFileBankDataOpen):Byte; virtual;
```

Funkce otevře požadovanou banku souborů pro přenos dat z nadřízeného systému do řídicího systému. Hodnota požadované banky je uložena v položce m_bFileNo.

Před voláním této funkce se provede konverze označení banku souboru pomocí volání fnConvertFileNo a nastaví se položka m_bFile. Poté se provede kontrola této položky na max. počet všech bank souborů. Pokud je tento počet překročen, vrací se chybový kód – erc_file_CannotBeExecuted.

Pokud banku souborů nelze otevřít, vrací se chybová hodnota, která je různá od erc_file_Ok. V opačném případě se vrací erc_file_Ok. V těle funkce by se měla porovnat velikost aktuální banky souborů s velikostí souboru, který se do banky bude zapisovat. Pokud se zjistí, že soubor do banky nelze zapsat, měla by se vracet chybová hodnota erc_file_InsufficientMemory.

Po volání této funkce se nastaví položka m_liFileOffs na 0.

6.5.3.5.10. fnPutBankDataBlock

```
function fnPutBankDataBlock
        (var FileBankDataBlock:TDataBlockFile):Byte; virtual;
```

Metoda zapíše datový buffer FileBankDataBlock.Block do aktuální banky souborů, která je nastavena v položce m_bFileNo s použitím dalších položek v proměnné FileBankDataBlock. Položka m_liFileOffs se může použít jako pomocná proměnná, ve které je uložena poslední pozice zapisovaných binárních dat.

Pokud lze blok dat bez chyby zapsat do aktuální banky souborů, vrací se erc_file_Ok. V opačném případě se vrací hodnota různá od erc_file_Ok.

6.6. Knihovna WTArch

Knihovna obsahuje implementaci funkčního objektu TWinTickArchive, který umožňuje přenos dat (záznamů) z archivů v řídicím systému do nadřízeného počítače. V manuálu archivů naleznete jejich podrobný popis, včetně vysvětlení práce s popisovači tabulek (deskriptorů) a popisovači grafů (rozšířených popisovačů).

6.6.1. Konstanty

Chybové kódy

`close_err_Abstract = 6;`

Při obsluze přijaté zprávy se volala funkce objektu TWinTickArchive.

`close_err_Param = 5;`

Přijatá zpráva obsahuje neplatné vstupní parametry – např. neplatný identifikátor archivů.

`close_err_Request = 4;`

Přijatá zpráva nemůže být obsloužena v daném stavu komunikačního automatu.

`close_err_Porus = 2;`

Při ukončování práce s archivem se zjistilo, že při přenosu dat došlo k přepsání dat. Proto nemůže být zajištěna jejich konzistence.

`close_err_Break = 1;`

Obsluha přijaté zprávy byla předčasně ukončena.

`close_err_OK = 0;`

Obsluha přijaté zprávy byla úspěšně dokončena.

6.6.2. Typy

```
TArchiveState = (
 as_IdleState,              { klidovy stav }
 as_ListOpen,               { otevreni seznamu archivu }
 as_ListItem,               { polozka ze seznamu archivu }
 as_ExListItem,             { rozsirena polozka ze seznamu archivu }
 as_ListClose,              { uzavira seznam archivu - pote prechazi
                           do klidoveho stavu }
 as_ArchiveOpen,            { otevreni vybraneho archivu }
 as_DataBlock               { blok dat z vybraneho archivu }
     );
```

Výčtový typ popisující stavy komunikačního automatu pro obsluhu požadavků spojených s přenosem dat z archivu v řídicím systému do nadřízeného systému.

Popisované typy jsou definovány v jednotce XTKing_1.pas. V této jednotce jsou definovány konstanty a typy, které se sdílí mezi vizualizací a řídicím systémem.

Popis typů je uveden pro úplnost uživatelského rozhraní.

```
TDataBlockArch = record
 NextBlockFl                : Boolean;    { nasleduje dalsi blok
                             prenasenych dat }
 DataFileOffs               : LongInt;    { offset prenasenych dat
                             banky archivu }
 BlockSize                  : Word; { velikost bloku prenasenych dat
                             banky archivu }
 Block                      : TBinBlockArch; { array [0..511] of
                             byte }
                             { datovy blok }
end;
```

Struktura slouží pro čtení dat v bance archivu.

```
TRecArchiveState = record
 UpDataFlg                  : Boolean;    { priznak noveho zaznamu od
                             posledniho cteni dat z archivu }
 DataLost                   : Boolean;    { priznak ztraty dat v
                             dusledku preteceni bufferu }
 TimeLastClear              : LongInt;    { (MS-DOS PackTime) cas
                             nejstarsiho prvku v archivu }
 TimeLastUpDate             : LongInt;    { (MS-DOS PackTime) cas
                             nejmladsiho prvku v archivu }
 ArchiveLength              : LongInt;    { delka zaplneneho bufferu
                             banky archivu }
end;
```

Struktura slouží pro zjištění aktuálního stavu banky archivu.

```
TArchiveRecDscr = record
 RecLen                     : Word; { udava delku nasledujici polozky
                             (masky zaznamu) v archivu }
 ResDscrMask                : array[0..MaxArchiveRecSize-1] of byte;
                             { 512B }
                             { DscrMsk_XXX – skutečná délka je daná
                             pomocí RecLen }
end;
```

Struktura slouží pro popis datový položek jednoho záznamu v archivu a slouží pro popis tabulky. Dále je tato struktura nazývána popisovačem tabulky.

```
TArchDataHeader = record
 NextHeaderFl               : Boolean;    { nasleduje dalsi Header }
 ExHeaderCount              : Byte; { pocet naslednych
                             TArchDataExHead }
 ArchiveId                  : Byte; { identifikace typu archivu -
                             ArchId }
 ArchBankNum                : Byte; { cislo banky archivu }
 ArchName                   : TArchName; { string[20] }
                             { jmeno banku archivu}
 ArchAttr                   : Byte; { atributy banky archivu –
                             ArchAttr }
 ArchState                  : TRecArchiveState;
                             { cislo banky archivu }
 ArchRecHeader              : TArchiveRecHeader;
                             { popisky/zahlavi polozek zaznamu }
 ArchFmtHeader              : TArchiveFmtHeader;
                             { format % strings polozek zaznamu like
                             CSV }
 ArchRecDscr                : TArchiveRecDscr;
                             { delka ovlivnena ArchivRecDscr.RecLen
                             !!!}
end;
```

Struktura obsahuje informace o struktuře archivu. Velikost struktury se mění podle velikosti položky ArchRecDscr a slouží k popisu tabulky.

```
TArchExRecDscr = record
 ExFieldAttrType            : Byte; { typ extended descriptoru }
 ExFieldSize                : Word; { velikost polozky - exfat_XX }
 ExFieldsCount              : Word; { skutecny pocet polozek v
                            ExFieldsAttrRec }
 ExFieldsAttrRec            : array[0..MaxExFieldsAttrRecSize-1] of
                            Byte;
                            { skutecna velikost je dana ExFieldSize
                            a ExFieldsCount }
end;
```

Struktura slouží pro popis zobrazení dat (záznamu) archivu v grafu.

```
TArchDataExHeader = record
 NextExHeaderFl             : Boolean;    { nasleduje dalsi ExHeader
                            }
 ArchExRecDscr              : TArchExRecDscr;
                            { promenna velikost dle pole
                            ExFieldsAttrRec }
end;
```

Struktura popisuje zobrazeni dat (záznamu) v grafu včetně příznaku zda bude následovat další hlavička.

Dále popisovaný typ je definován v jednotce ChnSofT.pas. V této jednotce jsou definovány konstanty a typy, které používá komunikační knihovna ChnSofs2.

Popis typu je zde uveden pro úplnost uživatelského rozhraní.

```
pMessDF1 = ^tMessDF1;
tMessDF1 = record
 Cmd        : Byte;         { identifikace zpravy }
 Rec        : tMess;        { vlastni prenasena data }
end;
```

6.6.3. Objekt TWinTickArchive

Objekt implementuje funkce pro vyčítání datových archivů.

6.6.3.1. Položky

`m_pAck : pARCH_ACK;`

Ukazatel na buffer vysílané zprávy potvrzující úspěšné provedení přijaté zprávy.

`m_pErr : pARCH_ERR;`

Ukazatel na buffer vysílané zprávy signalizující chybu při zpracování přijaté zprávy.

`m_pCmd : pARCH_CMD;`

Ukazatel na buffer s přijatou zprávou, který je přetypován na strukturu obsahující požadovanou obsluhu s archivem.

`m_AComState : tArchiveState;`

Proměnná obsahuje stav automatu pro vysílání dat archivu.

`m_bArchive : Byte;`

Index právě používané banky archivu.

`m_bExItem : Byte;`

Proměnná obsahuje index posledního použitého popisovače grafu (rozšířeného deskriptoru).

`m_liDataOffs : LongInt;`

Datový offset přenášených u jedné položky archivu.

6.6.3.2. Obecné metody

6.6.3.2.1. Konstruktor Init

`constructor Init;`

Konstruktor slouží k dokončení inicializace objektu s virtuálními metodami a jeho položek. V těle metody se volá konstruktor předka, pak se nastaví položky m_pAck, m_pErr a m_pCmd na NIL. Položky m_bBank, m_bExItem a m_liDataOffs se nastaví na 0. Položka m_AComState se nastaví na as_IdleState.

6.6.3.2.2. Destruktor Done

`destructor Done; virtual;`

Destruktor slouží ke zrušení objektu. V těle metody se volá destruktor předka.

6.6.3.3. Metody rozhraní s komunikačním objektem

6.6.3.3.1. Attach

`procedure Attach(pRMess:pMessDF1; pSMess:pMessDF1); virtual;`

Metoda, která se musí volat před zařazením funkčního objektu do smyčky zpracovávající přijaté zprávy. V těle metody se zavolá Attach předka, pak se nastaví položky m_pAck, m_pErr na datovou část vysílacího bufferu a m_pCmd na datovou část přijímacího bufferu komunikačního kanálu. Nakonec jsou položky m_bBank, m_bExItem a m_liDataOffs nastaveny na 0 a položka na m_AComState je nastavena na as_IdleState.

6.6.3.3.2. Detach

`procedure Detach; virtual;`

Metoda, která se automaticky volá po volání funkce UnRegister objektu TWinCom. V těle této metody se provádí deinicializace komunikačních bufferů.

V těle metody se nastaví m_pAck, m_pErr a m_pCmd na NIL a pak se zavolá Detach předka.

6.6.3.3.3. GetId

`function GetId:Byte; virtual;`

Funkce vrací identifikaci objektu. Objekt vrací wt_Archive.

6.6.3.3.4. Tick

`function Tick:Boolean; virtual;`

Funkce provádí zpracování přijaté zprávy a připravení odpovědi na tuto zprávu. Funkce vrací TRUE, pokud přijatá zpráva byla obsloužena. V opačném případě se vrací FALSE. Podrobnější popis zpracování přijatých zpráv najdete v kapitole 5.1.

V těle této funkce je implementován komunikační automat s obsluhou příkazů potřebných pro přenos dat archivů v řídicím systému. V jednotlivých stavech automatu se volají funkce s prefixem fn, dále funkce uživatelského rozhraní objektu. Tyto funkce musí být v potomku objektu předefinovány. Ve většině případů se doplní pouze funkční tělo funkce. (viz dodávané příklady)

6.6.3.3.5. SetResultCode

```
procedure SetResultCode(resCode:Word); virtual;
```

Metoda nastaví chybový návratový kód odpovídající obsluze přenosu dat archivu v řídicím systému. Pro odpověď se používají zprávy typu wcmd_ARCH_ERR.

6.6.3.4. GetMessSize

```
function GetMessSize:Word; virtual;
```

Funkce vrací délku zprávy připravené k odvysílání. (Pozn. Na rozdíl od předchozích verzí (do verze 1.02) tato délka neobsahuje velikost hlavičky protokolu DF2.)

Objekt implementuje výpočet délky zprávy typu wcmd_ARCH_ACK, wcmd_ARCH_ERR a pro ostatní typy zpráv volá GetMessSize předka.

6.6.3.5. Metody uživatelského rozhraní objektu

Následující metody musí být předefinovány a v jejich popisu bude popsán funkční kód, který se do nich musí doplnit.

6.6.3.5.1. fnConvertArchiveNo

```
function fnConvertArchiveNo(bBankNo:Byte):Byte; virtual;
```

Funkce vrací index banky archivu podle předaného označení banky archivu. Index archivu musí být vždy mezi [0..n-1].

V případě, že označení banky archivu je stejné jako index banky nemusí se tato funkce předefinovat.

Počet všech bank archivů lze získat pomocí funkce fnGetBankCnt.

6.6.3.5.2. fnGetArchiveCnt

```
function fnGetArchiveCnt:Byte; virtual;
```

Funkce vrací počet všech bank archivů v řídicím systému.

6.6.3.5.3. fnGetExDscrCnt

```
function fnGetExDscrCnt:Byte; virtual;
```

Funkce vrací počet popisovačů grafu. Tyto popisovače doplňují popisovače tabulky a jsou používány při zobrazování grafu.

6.6.3.5.4. fnAbort

```
procedure fnAbort;virtual;
```

Metoda provede kroky pro okamžité ukončení práce s archivy bez ohledu na současný stav komunikace. Povel se posílá jako reakce na uživatelský vstup – stisk tlačítka Abort.

Po volání této metody se nastaví položka m_bArchiveNo na 0 a položka m_AComState na as_IdleState.

6.6.3.5.5. fnClose

```
function  fnClose:Boolean; virtual;
```

Funkce provede kroky potřebné k ukončení práce s archivy. Při ukončování práce s archivy se provede kontrola současného stavu komunikace a pokud nedošlo k chybě, vrací se TRUE. Pokud došlo k chybě, vrací se FALSE.

Po volání této metody se nastaví položka m_bArchiveNo, m_liDataOffs na 0 a položka m_AComState na as_IdleState.

6.6.3.5.6. fnGetListOpen

```
procedure fnGetListOpen(var ArchListInfo:TArchListInfo); virtual;
```

Metoda vrací v položkách parametru ArchListInfo počet bank archivů (ArchListCount), sériové číslo řídicího systému (ControllerSN) a identifikační řetězec řídicího systému (ControllerIDStr). Počet bank archivů v řídicím systému se získá pomocí fnGetArchiveCnt, proto není potřeba tuto funkci předefinovat.

Po volání této metody se nastaví položka m_bArchiveNo na 0 a m_AComState na as_ListOpen.

Před voláním této funkce se kontroluje požadovaný typ archivu. Pokud typ archivu je neplatný, vrací se chybový kód close_err_Param.

6.6.3.5.7. fnGetListItem

```
procedure fnGetListItem
        (var ArchListItem:TArchDataHeader); virtual;
```

Metoda vrací v parametru ArchListItem informace o aktivní bance archivu. Informace o aktivní bance archivu je uložena v položce m_bArchiveNo, která se automaticky inkrementuje v komunikačním automatu po volání této funkce.

Před voláním této funkce se kontroluje položka m_bArchiveNo na max. počet všech bank archivů. Pokud je tento počet překročen, vrací se chybový kód – close_err_Param. Dále je kontrolován současný stav komunikačního automatu. V případě neplatného stavu se vrací close_err_Request.

Nastavení dalšího stavu komunikačního automatu závisí na tom, zda archiv podporuje grafy. Pokud jsou grafy podporovány, nastaví se m_AComState na as_ExListItem. V opačném případě se nastaví m_AComState na as_ListItem.

6.6.3.5.8. fnGetListExItem

```
procedure fnGetListExItem
        (var ArchListExItem :TArchDataExHeader); virtual;
```

Metoda nastaví strukturu popisovače grafu podle aktuálně vybraného popisovač v položce m_ExItem pro aktuální banku archivu m_bArchive.

Před voláním této funkce se kontroluje položka m_bExItem na max. počet všech popisovačů grafu. Pokud je tento počet překročen vrací se close_err_Param. Dále je kontrolován současný stav komunikačního automatu. V případě neplatného stavu se vrací close_err_Request.

Nastavení dalšího stavu komunikačního automatu závisí na tom, zda se bude ještě vysílat další popisovač grafu. V případě, že se další popisovač nebude vysílat, nastaví se m_AComState na as_ListItem. Jinak nastavení m_AComState zůstane stejné, tj. as_ExListItem.

6.6.3.5.9. fnGetState

```
procedure fnGetState
        (var ArchiveRecState:TRecArchiveState); virtual;
```

Metoda nastaví aktuální stav vybrané banky archivu do parametru ArchiveRecState. Výběr banky archivu je určen pomocí položky m_bArchive.

Před voláním funkce se provede konverze označení banku na index banku pomocí funkce fnConvertArchiveNo. Dále se provede kontrola hodnoty aktivního banku a typu archivu. Jeli jedna z kontrol neplatná vrací se close_err_Param.

Položka m_AComState se nastaví na as_IdleState.

6.6.3.5.10. fnOpenArchive

```
procedure fnOpenArchive(BegRecTime, EndRecTime:LongInt; var Prvni,
          Posledni, Length: LongInt); virtual;
```

Metoda otevře požadovanou banku archivu pro čtení, její index je nastaven v položce m_bArchive. Procedura vrací v proměnných *Prvni* a *Posledni* čas záznamů, které lze z archivu přečíst na základě požadovaného časového intervalu. Časový interval se zadává pomocí proměnných BegRecTime a EndRecTime. Dalším výstupem je celková délka dat (záznamů), které lze z archivu v čase volání této procedury přečíst.

Pozn. Celková délka přečtených dat (záznamů) se může lišit, protože může dojít k jejich přepisu.

Před voláním funkce se provede konverze označení banku na index banku pomocí funkce fnConvertArchiveNo a tato hodnota se uloží do položky m_bArchive. Poté se provede kontrola této položky na max. počet všech bank archivů a typu archivu. Jeli jedna z kontrol neplatná, vrací se close_err_Param.

Položka m_AComState se nastaví na as_ArchiveOpen.

6.6.3.5.11. fnReadArchive

```
function  fnReadArchive(pData:Pointer; EndTime:LongInt;
          var Len:Word):Boolean; virtual;
```

Metoda nastaví do datového bufferu pData jeden záznam podle stavových proměnných otevřeného archivu m_bArchive. Při návratu se vrací délka nastaveného datového bufferu a příznak, zda lze ještě z bufferu číst další záznam.

Pozn. V dalším cyklu se může stát, že z archivu už nelze číst záznam v důsledku jejich přepsání – přetečení archivu.

Před voláním funkce se provede kontrola stavu automatu. Pokud je v položce m_AComState nastaven neplatný stav, vrací se close_err_Request.

Položka m_AComState se nastaví na as_DataBlock.

6.6.3.5.12. fnClearArchive

```
function  fnClearArchive:Boolean; virtual;
```

Funkce smaže vybranou banku archivu. Index vybrané banky je nastaven v položce m_bArchive. Pokud při mazání banky archivu nedojde k žádné chybě, vrací se True. V opačném případě se vrací False.

Před voláním funkce se provede konverze označení banku na index banku pomocí funkce fnConvertArchiveNo a tato hodnota se uloží do položky m_bArchive. Poté se provede jejich kontrola na max. počet všech bank archivů a typu archivu. Jeli jedna z kontrol neplatná, vrací se close_err_Param.

Položka m_AComState se nastaví na as_IdleState.