Valids

KNIHOVNA VALIDÁTORŮ PRO VIZUALIZACE NA JEDNOTCE 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 1.00

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: 21.01.2004
Datum posledního uložení dokumentu: 21.01.2004
(Datum vydání a posledního uložení dokumentu musí být stejné)

Obsah :

1. O dokumentu

1.1. Revize dokumentu

Verze dokumentu	Verze SW	Autor	Datum vydání	Popis změn
1.00	1.00	Cr	21.01.2004	První vydání

1.2. Účel dokumentu

Tento dokument slouží jako popis knihovny Valids, která je součástí balíku vizualizačních knihoven pro jednotku KIT.

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

Knihovna validátorů je podpůrná knihovna vizualizačního systému umožňující snadnou editaci jednoduchých datových typů. Obsahuje množství tříd pro editaci různých datových typů, jako je Byte, Integer, Real, String apod. Validátory jsou určeny pro přímou spolupráci s komponentami vizualizačního systému, především s komponentou **TEdit**, příp. i dalšími komponentami jako je **TTrackBar**, **TScrollBar**, **TUpDown**, **TStaticText** apod.

3.1. Funkce a třídy validátorů

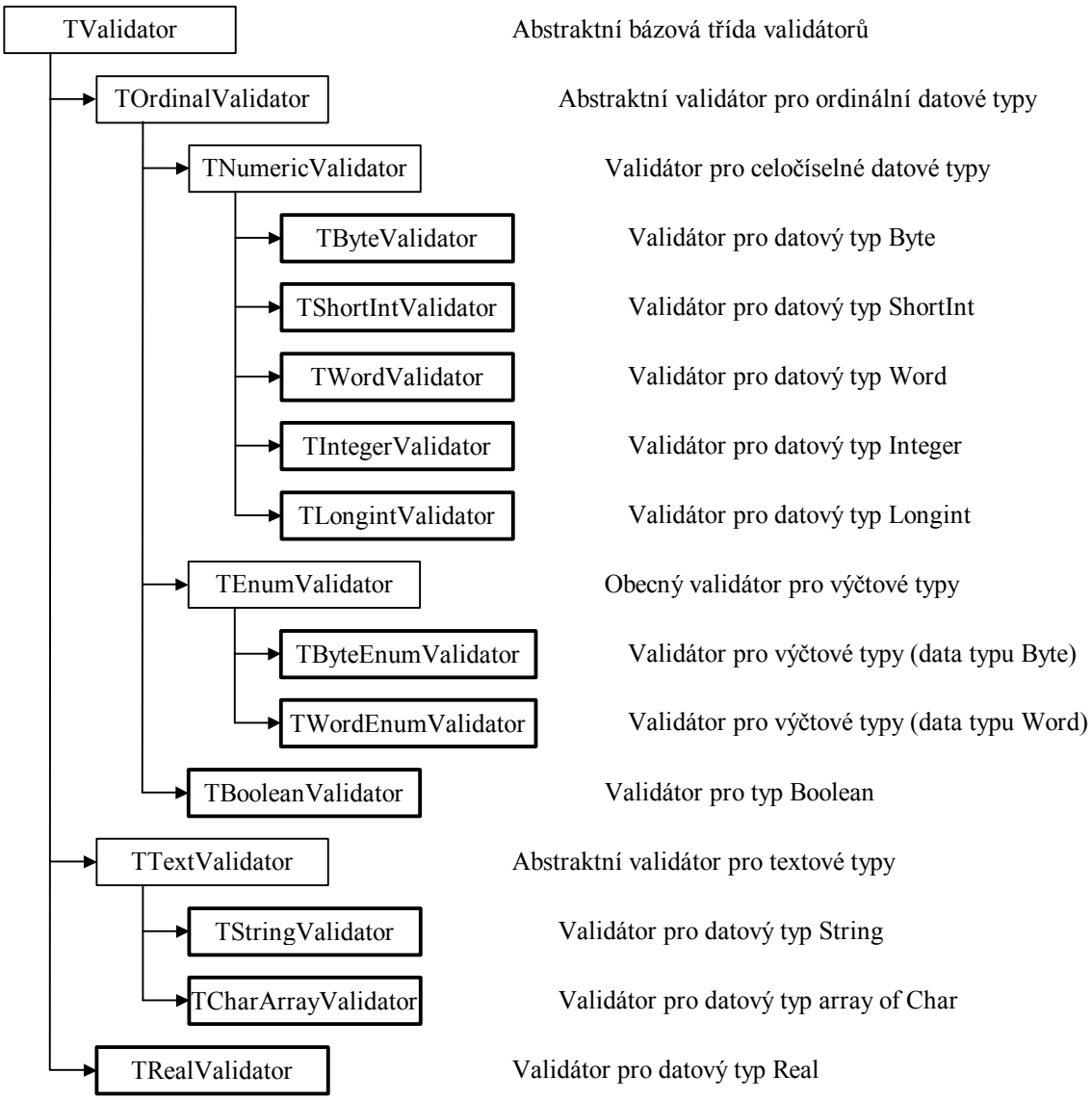
Samotný validátor je třída, která provádí tři hlavní činnosti:

- formátování binární hodnoty na textový řetězec
- formátování textového řetězce na binární hodnotu
- kontrolu syntaxe a sémantiky textového řetězce

Dále obvykle každý validátor:

- umožňuje filtrování zadávaných znaků v komponentě TEdit
- vrací standardní text chybového hlášení, při neúspěšné validaci zadané hodnoty

Všechny validátory vycházejí z abstraktní třídy TValidator (viz. kapitola 4.3.1). Viz. strom tříd validátoru na následujícím obrázku:



3.2. Společné metody validátorů

Každá instance validátoru si pamatuje ukazatel do paměti, kde je uložena kopie editovaných dat. Ukazatel na tato data je možné získat pomocí metody **GetDataPtr** (viz. kapitola 4.3.1.3).

```
function  TValidator.GetDataPtr: Pointer; virtual;
```

Počet bajtů editovaného datového typu je možné zjistit pomocí metody **GetDataSize** (viz. kapitola 4.3.1.6)

```
function  TValidator.GetDataSize: Word; virtual;
```

Pro manipulaci s daty svázanými s validátorem je určena metoda **TransferText** (viz. kapitola 4.3.1.6).

```
function  TValidator.TransferText( AText: PString; AMaxLen: Integer;
```

```
            AMode: Integer ): Integer; virtual;
```

Metoda **TransferText** na základě parametru AMode provádí převod binárních dat na textový řetězec, převod textového řetězce na binární data a zároveň provádí kontrolu syntaxe a sémantiky tohoto textového řetězce. Metoda vrací jeden z návratových kódu s prefixem ve_ (viz. kapitola 4.1.2)

Validátory odvozené ze třídy **TOrdinalValidator** nabízejí jako variantu k metodě **TrasferText** metodu **TransferOrdinal**.

```
function  TOrdinalValidator.TransferOrdinal( var AValue: Longint;
            AMode: Integer ): Integer; virtual;
```

Metoda **TransferOrdinal** neprovádí převod binární hodnoty na textový řetězec, pouze převádí binární hodnotu na typ Longint a kontroluje pouze sémantiku této hodnoty.

```
function TOrdinalValidator.GetMaxTextLength: Integer; virtual;
```

Metoda **GetMaxTextLength** vrací maximální počet znaků potřebný pro editaci hodnoty v textové podobě. Metoda slouží k automatizaci editace u speciálních komponent jako je **TListEdit** apod.

Pro filtrování znaků, které může editovaný textový řetězec reprezentující validovanou hodnotu obsahovat slouží metoda **CharFilter** (viz. kapitola 4.3.1.5)

```
function  TValidator.CharFilter( Ch: Char ): Boolean; virtual;
```

Metoda **CharFilter** je volána, např. z komponenty **TEdit**, při stisku klávesy reprezentující znak. Pokud je znak platný vrací hodnotu True.

V případě, že validace hodnoty neproběhne úspěšně, vrací metoda **TransferText** příp. **TransferOrdinal** chybový kód, tj. jednu z konstant s prefixem ve_ (viz. kapitola 4.1.2). Pro usnadnění výpisu případného chybové hlášení pro uživatele, poskytuje každý validátor metodu **GetErrorStr** (viz. kapitola 4.3.1.8).

```
procedure GetErrorStr( AErrCode: Integer; var AErrStr: string );
```

Metoda **GetErrorStr** na základě chybového kódu vráceného jednou z metod **TransferXXX** vytvoří textový řetězec popisující chybu, který je možné vypsat např. v dialogovém okně apod.

3.3. Spolupráce validátorů a komponent

Validátory úzce spolupracují s komponentami. Každý potomek třídy **TControl** poskytuje následující podporu pro validaci vlastní hodnoty:

Položka Validator obsahuje ukazatel na validátor svázaný s komponentou.

Validator : PValidator;

Položku Validator je potřeba inicializovat hned po vytvoření instance komponenty, viz. následující příklad:

```
var
  MyInt : Integer;
  Edit  : TEdit;

Edit := New( PEdit, Init( Bounds, 20 ));
Edit^.Customize( ccEdit );
Edit^.Validator := New( PIntegerValidator,
                    Init( @MyInt, nvfRadixDec, 0, 100 ));
```

Výše uvedený kódem se instanci komponenty TEdit přiřadí validátor datového typu Integer. Edit se bude decimální obraz hodnoty v rozsahu 0 až 100.

Dále každá komponenta poskytuje virtuální metodu Transfer, jež plní tři následující funkce:

- přesun hodnoty z validátoru do komponenty
- validace hodnoty komponenty pomocí validátoru
- přesun hodnoty z komponenty do validátoru

Metoda je definována následovně:

```
function TControl.Transfer( AMode: Integer ): Integer; virtual;
```

Parametr AMode udává, kterou ze třech výše uvedených funkci má metoda provést. Může nabývat jedné ze třech hodnot:

vmLoad	Načtení dat z validátoru do komponenty
vmStore	Uložení hodnoty komponenty do dat validátoru
vmValidate	Kontrola správnosti hodnoty komponenty.

Metoda Transfer vrací chybový kód, tj. jednu z konstant s prefixem ve_ (viz. kapitola 4.1.2)

Tuto metodu musí potomci třídy **TControl** předefinovat, tak aby správně prováděla výše uvedené tři funkce. Přímo ve třídě **TControl** je metoda Transfer implementovaná tak, že vrací vždy hodnotu veOk a neprovádí žádnou akci.

Každá komponenta poskytuje dále metodu **Validate**, která provádí samotnou validaci hodnoty komponenty pomocí vlastního validátoru.

```
function TControl.Validate: Boolean;
```

Metoda **Validate** volá interně metodu Transfer s parametrem vmValidate. Pokud validace proběhne úspěšně metoda vrátí hodnotu True. V opačném případě je zavolána metoda **TControl.Error**, která implicitně vyvolá oznámení **nmError** (více v dokumentaci ke knihovně Controls).

3.4. Použití validátoru

Metody komponent **Transfer** a **Validate** je možné volat v libovolném okamžiku ručně. Volání metody **Control^.Transfer(vmLoad)** způsobí přepis hodnoty z validátoru do komponenty. Volání metody **Control^.Transfer(vmStore)** způsobí přepis hodnoty komponenty do validátoru. Volání metody **Validate**, způsobí kontrolu hodnoty komponenty a případné vyvolání oznámení nmError.

Pokud je zavolána metoda **Transfer** nebo **Validate** u potomka komponenty **TGroup** (např. **TDialog**, **TPage** apod), pak se provede přesun, příp. validace hodnoty u všech komponent vložených do této skupiny, které mají přiřazen vlastní validátor.

Při volání metody **TGroup.ExecControl** jsou metody **Transfer** a **Validate** volání automaticky. Metoda **ExecControl** volá nejprve metodu **Transfer** s parametrem vmLoad. Před ukončením modálního stavu, je zavolána metoda **Validate**. Modální stav je ukončen pouze tehdy, jestliže tato metoda vrátí hodnotu True, příp. pokud je modální stav ukončován s hodnotou jinou než mrOk. Po návratu z metody **Execute** příslušné komponenty je otestován výsledek modálního stavu, pokud je roven hodnotě mrOk, pak je automaticky zavolána metoda **Transfer** s parametr vmStore.

4. Reference

4.1. Konstanty

4.1.1. Konstanty vm_

Konstanty s prefixem vm_ určují chování metody **TranfersText** (viz. kapitola 4.3.1.6) resp. **TransferOrdinal** (viz. kapitola 4.3.2.5). Tyto konstanty jsou zároveň parametrem metody **Transfer** třídy **TControl**.

Identifikátor	Kód	Význam
vmLoad	0	Načtení binární hodnoty z ukazatele vráceného metodou **GetDataPtr**. Konverze této hodnoty na textový řetězec (resp. převod na typ Longint).
vmStore	1	Konverze textového řetězce (resp. typu Longint) na binární hodnotu a její uložení na místo vrácené metodou **GetDataPtr**. Při konverzi je zároveň provedena validace.
vmValidate	2	Validace textového řetězce (resp. typu Longint), bez uložení.

4.1.2. Konstanty ve_

Konstanty s prefixem ve_ jsou návratové kódy metod **TranfersText** (viz. kapitola 4.3.1.6) resp. **TransferOrdinal** (viz. kapitola 4.3.2.5).

Identifikátor	Kód	Význam
veOk	0	Metoda proběhla úspěšně.
veNotImpl	1	Funkce metody není implementovaná.
veSyntax	2	Neplatná syntaxe validovaného řetězce.
veRange	3	Neplatný rozsah hodnoty.
veLength	4	Neplatná délka řetězce.

4.1.3. Konstanty ovf_

Konstanty s prefixem ovf_ (ordinal validator flag) upřesňují vlastnosti dat validátoru **TOrdinalValidator** a jeho potomků. Kombinace těchto konstant je uložena v položce DataFlags třídy **TOrdinalValidator**.

Identifikátor	Kód	Význam
ovfSizeMask	$03	Maska pro příznaky délky datového typu
ovfSizeByte	$00	8bitový datový typ
ovfSizeWord	$01	16bitový datový typ
ovfSizeLong	$02	32bitový datový typ
ovfSigned	$04	Datový typ se znaménkem

4.1.4. Konstanty nvf_

Konstanty s prefixem nvf_ (numeric validator flag) upřesňují vlastnosti validátoru **TNumericValidator** a jeho potomků. Kombinace těchto konstant je uložena v položce FmtFlags třídy **TNumericValidator**.

Identifikátor	Kód	Význam
nvfRadixMask	$03	Maska pro příznaky
nvfRadixDec	$00	Decimálního zápis hodnoty.
nvfRadixBin	$01	Binární zápis hodnoty.
nvfRadixHex	$02	Hexadecimální zápis hodnoty.
nvfFixedPoint	$03	Číslo s pevné řádovou čárckou.
nvfCharFilter	$04	Zapnutí filtrování znaku metodou **CharFilter**.

4.1.5. Konstanty tvf_

Konstanty s prefixem tvf_ (text validator flag) upřesňují chování validátoru **TTextValidátor** a jeho potomků. Kombinace těchto konstant je uložena v položce FmtFlags třídy **TTextValidator**.

Identifikátor	Kód	Význam
tvfCheckLength	$01	Kontrola délky zadaného řetězce. Pokud je nastaven tento příznak, vrací funkce TransferText chybový kód veLength, pokud je překročena délka zadaného řetězce. Pokud tento příznak nastaven není, pak je řetězec

		oříznut zprava na požadovanou délku.
tvfTrimLeft	$02	Pokud je nastaven tento příznak, budou u řetězce před uložením ořízнuty všechny mezery zleva.
tvfTrimRight	$04	Pokud je nastaven tento příznak, budou u řetězce před uložením ořízнuty všechny mezery zprava.
tvfUpperCase	$08	Pokud je nastaven tento příznak, bude řetězec před uložením převeden na velká písmena.
tvfLowerCase	$10	Pokud je nastaven tento příznak, bude řetězec před uložením převeden na malá písmena.

4.1.6. Konstanty rvf_

Konstanty s prefixem rvf_ (real validator flag) upřesňují chování validátoru **TRealValidator**. Kombinace těchto konstant je uložena v položce FmtFlags třídy **TRealValidator**.

Identifikátor	Kód	Význam
rvfCharFilter	$01	Zapnutí filtrování znaků metodou **CharFilter**.

4.2. Typy

4.2.1. Typ TEnumLookupFunc

Typ **TEnumLookupFunc** definuje prototyp funkce vracející ukazatel na řetězec reprezentující zadanou ordinální hodnotu vyčtového typu. Tato funkce se využívá ve validátorech výčtových typů (viz. **TEnumValidator**, **TBooleanValidator**)

```
TEnumLookupFunc = function( AValue: Longint ): PString;
```

4.3. Třídy

4.3.1. Třída TValidator

Abstraktní třída **TValidator** je bázovou třídou definující základní rozhraní všech validátorů. Potomci této třídy musí předefinovat minimálně metody **GetDataSize** a **TransferText**.

```
PValidator = ^TValidator;
TValidator = object( TObject )
public
  DataPtr : Pointer;

  constructor Init( ADataPtr: Pointer );
  function  GetDataPtr: Pointer; virtual;
  function  GetDataSize: Word; virtual;
  function  CharFilter( Ch: Char ): Boolean; virtual;
  function  TransferText( AText: PString; AMaxLen: Integer;
              AMode: Integer ): Integer; virtual;
```

```
    function  GetMaxTextLength: Integer; virtual;
    procedure GetErrorStr( AErrCode: Integer; ¨
                var AErrStr: string ); virtual;
  end;
```

4.3.1.1. Položka TValidator.DataPtr

Položka **DataPtr** obsahuje ukazatel na binární podobu validovaných dat. Položka je inicializována konstruktorem třídy a je určena pouze pro čtení. Potomci třídy **TValidator** by měli vzhledem k dalším rozšiřováním možností validátoru využít pro čtení této položky metodu **GetDataPtr** (viz. kapitola 4.3.1.3).

```
DataPtr : Pointer;
```

4.3.1.2. Konstruktor TValidator.Init

Konstruktor **Init** provádí inicializaci instance třídy.

```
constructor Init( ADataPtr: Pointer );
```

Parametry:

ADataPtr	Odkaz na validovaná data. Podle tohoto parametru je inicializován položka DataPtr.

Návratové hodnoty:

Konstruktor nevrací žádnou hodnotu.

Poznámky:

4.3.1.3. Metoda TValidator.GetDataPtr

Metoda **GetDataPtr** vrací ukazatel na binární podobu validovaných dat.

```
function  GetDataPtr: Pointer; virtual;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda vrací ukazatel na binární podobu validovaných dat, určenou při volání konstruktoru instance.

Poznámky:

Metoda **GetDataPtr** vrací implicitně hodnotu položky DataPtr.

4.3.1.4. Metoda TValidator.GetDataSize

Abstraktní metoda **GetDataSize** vrací délku binárních dat na které ukazuje ukazatel vrácený metodou **GetDataPtr**.

```
function  GetDataSize: Word; virtual;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda vrací délku dat v bajtech.

Poznámky:

Potomci třídy **TValidator** musí metodu **GetDataSize** předefinovat tak, aby vracela správnou délku validovaných dat.

4.3.1.5. Metoda TValidator.CharFilter

Metoda **CharFilter** určuje zda zadaný znak je platný v textové reprezentaci validovaných dat.

```
function  CharFilter( Ch: Char ): Boolean; virtual;
```

Parametry:

Ch Posuzovaný znak.

Návratové hodnoty:

Metoda vrací hodnotu True, pokud je zadaný znak platný. V opačném případě vrací hodnotu False.

Poznámky:

Přímo ve třídě **TValidator** je metoda **CharFilter** implementována tak, že vrací vždy hodnotu True. Potomci této třídou mohou metodu **CharFilter** předefinovat

Metodu **CharFilter** využívá např. komponenta **TEdit**, která povolí zadat pouze platné znaky. U ostatních komponent se metoda **CharFilter** většinou nevyužívá.

4.3.1.6. Metoda TValidator.TransferText

Metoda **TransferText** tvoří jádro validátoru. Provádí převod binární hodnoty svázané s validátorem na textový řetězec a naopak, dále provádí validaci tohoto textového řetězce.

```
function  TransferText( AText: PString; AMaxLen: Integer;
            AMode: Integer ): Integer; virtual;
```

Parametry:

AText	Ukazatel na buffer pro textový řetězec. Délka bufferu je dána parametrem AMaxLen. Tento buffer je určen buď pro zápis nebo pro čtení, podle nastavení parametru AMode.
AMaxLen	Délka bufferu pro textový řetězec, parametr AMaxLen je brán v potaz pouze tehdy, jestliže parametr AMode je nastaven na hodnotu vmLoad. V opačném případě může být hodnota tohoto parametru libovolná.
AMode	Určuje chování metody a může obsahovat tři různé hodnoty:

vmLoad	Metoda provede konverzi binární hodnoty, na kterou ukazuje funkce **GetDataPtr** na textový řetězec.
vmStore	Metoda provede validaci a konverzi textového řetězce na binární hodnotu, na kterou ukazuje funkce **GetDataPtr**.
vmValidate	Metoda provede pouze validaci textového řetězce.

Návratové hodnoty:

Metoda **TransferText** vrací jednu z konstant s prefixem ve_ (viz. kapitola 4.1.2). V případě úspěšného provedení vrací konstantu veOk.

Poznámky:

Metoda **TransferText** je abstraktní. Potomci třídy **TValidator** musí tuto metodu předefinovat.

V případě, že metoda **TransferText** skončí chybou, tj. vrátí návratový kód různý od veOk, je možné zavolat metodu **GetErrorStr** (viz. kapitola 4.3.1.8) s tímto návratovým kódem. Metoda **GetErrorStr** vytvoří textový řetězec popisující konkrétní typ chyby. Tento řetězec je možné zobrazit např. v dialogovém okně apod.

4.3.1.7. Metoda TValidator.GetMaxTextLength

Metoda **GetMaxTextLength** vrací maximální počet znaků potřebný pro editaci hodnoty v textové podobě.

```
function  GetMaxTextLength: Integer; virtual;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda vrací počet znaků.

Poznámky:

Metoda **GetMaxTextLength** je určena k automatizaci editací u speciálních komponent jako je např. **TListEdit**. Metoda vrací maximální počet znaků editovaného textu, který je nutný pro nastavení komponenty **TEdit**. Metoda **GetMaxTextLength** třídy **TValidator** vrací vždy hodnotu 255. Potomci třídy **TValidator** by měli tuto metodu předefinovat, tak aby vracela co nejmenší počet znaků, potřebný pro editaci vlastní hodnoty.

4.3.1.8. Metoda TValidator.GetErrorStr

Metoda **GetErrorStr** převádí chybový kód ve_ (viz. kapitola 4.1.2] na textový řetězec.

```
procedure GetErrorStr( AErrCode: Integer;
  var AErrStr: string ); virtual;
```

Parametry:

AErrCode	Chybový kód ve_ (viz. kapitola 4.1.2).
AErrStr	Buffer, do kterého bude uložen text popisující chybu danou parametrem AErrCode.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Přímo ve třídě **TValidator** je metoda **GetErrorStr** implementována tak, že nastaví parametr AErrStr vždy na prázdný řetězec. Potomci této třídy mohou metodu **GetErrorStr** předefinovat tak, aby vracela vhodnou textovou reprezentaci chybového kódu.

4.3.2. Třída TOrdinalValidator

Abstraktní třída **TOrdinalValidator** definuje rozhraní a pomocné metody pro validátory ordinálních typů, jako je výčtový typ, typ bajt, integer, longint apod. Potomci této třídy musí předefinovat minimálně metody **TransferText** a **GetMinMax**.

Třída **TOrdinalValidator** definuje novou metodu **TransferOrdinal**, která má podobné použití jako metoda **TransferText**. Metoda **TransferOrdinal** převádí binární podobu validovaných dat na typ Longint a naopak a provádí validaci dat, např. kontrolu rozsahu apod.

Některé typy komponent jako je např. **TScrollBar**, **TTrackBar**, **TUpDown** vyžadují

aby připojený validátor vycházel ze třídy **TOrdinalValidator**.

```
POrdinalValidator = ^TOrdinalValidator;
TOrdinalValidator = object( TValidator )
public
  DataFlags : Byte;

  constructor Init( ADataPtr: Pointer; ADataFlags: Byte );
  function  GetDataSize: Word; virtual;
  function  ValueFilter( AValue: Longint ): Integer; virtual;
  function  TransferOrdinal( var AValue: Longint;
              AMode: Integer ): Integer; virtual;
  procedure GetMinMax( var AMin, AMax: Longint ); virtual;
  procedure StoreValue( AValue: Longint );
  function  LoadValue: Longint;
  function  EnumLookup( AValue: Longint ): PString; virtual;
  function  NextValue( AValue: Longint;
              var AForward: Boolean ): Longint; virtual;
end;
```

4.3.2.1. Položka TOrdinalValidator.DataFlags

Položka **DataFlags** upřesňuje vlastnosti dat, na které ukazuje metoda **GetDataPtr**. Položka obsahuje kombinaci příznaků ovf_ (viz. kapitola 4.1.3). Položka DataFlags je inicializována parametrem konstruktoru třídy a je určena pouze pro čtení.

```
DataFlags : Byte;
```

4.3.2.2. Konstruktor TOrdinalValidator.Init

Konstruktor **Init** provádí inicializaci instance validátoru.

```
constructor Init( ADataPtr: Pointer; ADataFlags: Byte );
```

Parametry:

ADataPtr	Odkaz na validovaná data.
ADataFlags	Vlastnosti validovaných dat, resp. jejich datový typ (viz. příznaky ovf_ v kapitole 4.1.3).

Návratové hodnoty:

Konstruktor nevrací žádnou hodnotu.

Poznámky:

Parametr konstruktoru ADataPtr, společně s parametrem ADataFlags určují místo v paměti a datový typ (a tedy i délku dat). Viz následující tabulka:

DataFlags	Datový typ
ovfByte	Byte
ovfByte or ovfSigned	ShortInt
ovfWord	Word
ovfWord or ovfSigned	Integer

ovfLong or ovfSigned	Longint

4.3.2.3. Metoda TOrdinalValidator.GetDataSize

Metoda **GetDataSize** vrací délku binárních dat na které ukazuje ukazatel vrácený metodou **GetDataPtr**.

```
function  GetDataSize: Word; virtual;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda vrací délku dat v bajtech.

Poznámky:

Metoda **TOrdinalValidator.GetDataSize** předefinována abstraktní metodu **TValidator.GetDataSize** a vrací počet bajtů na základě obsahu položky DataFlags (viz. kapitola 4.3.2.1).

4.3.2.4. Metoda TOrdinalValidator.ValueFilter

Metoda **ValueFilter** slouží k validaci hodnoty, určuje zda je zadaná hodnota platná či nikoli.

```
function  ValueFilter( AValue: Longint ): Integer; virtual;
```

Parametry:

AValue Kontrolovaná hodnota.

Návratové hodnoty:

Metoda **ValueFilter** vrací chybový kód ve_ (viz. kapitola 4.1.2). V případě, že je zadaná hodnota v pořádku, vrací konstantu veOk.

Poznámky:

Metoda **ValueFilter** je volána v rámci metody **Transfer** s parametrem vmStore a vmValidate.

Třída **TOrdinalValidator** implementuje metodu **ValueFilter** tak, že zavolá metodu **GetMinMax**, zjistí minimální a maximální hodnotu, porovná ji se zadanou a v případě, že je hodnota mimo rozsah vrátí konstantu veRange.

Potomci třídy TOrdinalValidator mohou metodu ValueFilter rozšířit o další kontroly, viz. následující příklad:

```
const
  veAlign = 10;

function TMyOrdinalValidator.ValueFilter(
           AValue : Longint ): Boolean;
var
  Result : Integer;
begin
  Result := inherited ValueFilter( AValue );
  if Result = veOk then
  begin
    { V pripade, ze hodnota neni zarovnana na 256, vraci metoda
    { chybu veAlign }
    if AValue and 255 <> 0 then Result := veAlign;
  end;
  ValueFilter := Result;
end;
```

4.3.2.5. Metoda TOrdinalValidator.TransferOrdinal

Metoda **TransferOrdinal** je doplňkem k metodě **TransferText** (viz. kapitola 4.3.1.6). Provádí převod binární hodnoty svázané s validátorem na hodnotu typu Longint a naopak, dále provádí validaci této hodnoty.

```
function  TransferOrdinal( var AValue: Longint;
            AMode: Integer ): Integer; virtual;
```

Parametry:

AValue	Odkaz na proměnnou, která obsahuje nebo bude naplněna hodnotou. V případě, že parametr AMode je nastaven na hodnotu vmLoad funguje tento parametr jako výstupní, v ostatních případech se jedná o vstupní parametr.
AMode	Určuje chování metody a může obsahovat tři různé hodnoty:
vmLoad	Metoda provede převod binární hodnoty, na kterou ukazuje funkce **GetDataPtr** na hodnoty typu Longint.
vmStore	Metoda provede validaci a převod hodnoty typu Longint na binární hodnotu, na kterou ukazuje funkce **GetDataPtr**.
vmValidate	Metoda provede pouze validaci hodnoty typu Longint.

Návratové hodnoty:

Metoda **TransferText** vrací jednu z konstant s prefixem ve_ (viz. kapitola 4.1.2). V případě úspěšného provedení vrací konstantu veOk.

Poznámky:

V případě, že metoda **TransferText** skončí chybou, tj. vrátí návratový kód různý od

veOk, je možné zavolat metodu **GetErrorStr** (viz. kapitola 4.3.1.8) s tímto návratovým kódem. Metoda **GetErrorStr** vytvoří textový řetězec popisující konkrétní typ chyby. Tento řetězec je možné zobrazit např. v dialogovém okně apod.

4.3.2.6. Metoda TOrdinalValidator.GetMinMax

Metoda **GetMinMax** zjišťuje platný rozsah validované hodnoty.

```
procedure GetMinMax( var AMin, AMax: Longint ); virtual;
```

Parametry:

AMin	Odkaz na proměnnou, do které bude uložena minimální hodnota.
AMax	Odkaz na proměnnou, do které bude uložena maximální hodnota.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Třída **TOrdinalValidator** implementuje metodu **GetMinMax**, tak že vrací minimální a maximální hodnotu definovanou typem daným položkou DataFlags, viz. následující tabulka:

DataFlags	Min	Max
ovfByte	0	255
ovfByte or ovfSigned	-128	127
ovfWord	0	65535
ovfWord or ovfSigned	-32768	32767
ovfLong	- 2147483648	2147483647
ovfLong or ovfSigned	- 2147483648	2147483647

Potomci třídy **TOrdinalValidator** mohou metodu **GetMinMax** předefinovat tak, aby vracela jiný rozsah.

4.3.2.7. Metoda TOrdinalValidator.StoreValue

Metoda **StoreValue** uloží zadanou hodnotu na místo v paměti, na které ukazuje metoda **GetDataPtr**.

```
procedure StoreValue( AValue: Longint );
```

Parametry:

AValue	Ukládaná hodnota.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Pokud má parametr AValue hodnotu mimo rozsah datového typu definovaného položkou DataFlags, pak je ukládaná hodnota oříznuta na počet bitů daný tímto typem.

4.3.2.8. Metoda TOrdinalValidator.LoadValue

Metoda **LoadValue** přečte hodnotu v paměti z místa, na které ukazuje metoda **GetDataPtr**.

```
function  LoadValue: Longint;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda vrací hodnotu uloženou na místě v paměti, na které ukazuje metoda **GetDataPtr**.

Poznámky:

4.3.2.9. Metoda TOrdinalValidator.EnumLookup

Metoda **EnumLookup** je pomocná metodou pro podporu výčtových typů. Metoda vrací ukazatel na název varianty výčtového typu.

```
function  EnumLookup( AValue: Longint ): PString; virtual;
```

Parametry:

AValue	Ordinální hodnota příslušné varianty výčtového typu.

Návratové hodnoty:

Metoda vrací ukazatel na znakový řetězec pro příslušnou variantu výčtového typu. Pokud je parametr AValue mimo rozsah výčtového typu, nebo variantě odpovídá prázdný řetězec, metoda vrací hodnotu **nil**.

Poznámky:

Přímo ve třídě **TOrdinalValidator** je metoda **EnumLookup** implementována tak, že vrací ukazatel na znakový řetězec typy '#' + Str(AValue), tj. např. #0, #2, #5 atd. Potomci třídy **TOrdinalValidator** mohou tuto metodu předefinovat.

4.3.2.10. Metoda TOrdinalValidator.NextValue

Metoda **NextValue** je pomocnou metodou pro podporu výčtových typů. Metoda zjišťuje následníka příp. předchůdce zadané hodnoty výčtového typu.

```
function  NextValue( AValue: Longint;
  var AForward: Boolean ): Longint; virtual;
```

Parametry:

AValue	Ordinální hodnota varianty výčtového typu.
AForward	Parametr určuje, zda metoda vrátí následníka (True) nebo předchůdce (False).

Návratové hodnoty:

Metoda vrací ordinální hodnotu následníka, příp. předchůdce hodnoty zadané parametrem AValue.

Poznámky:

Přímo ve třídě **TOrdinalValidator** je metoda **NextValue** implementována tak, že následník resp. předchůdce má o 1 vyšší, resp. nižší hodnotu. Předchůdce minimální hodnoty je maximální hodnota. A následník maximální hodnoty je minimální hodnota. Minimální a maximální hodnota je určena metodou **GetMinMax** (viz. kapitola 4.3.2.6).

4.3.3. Třída TNumericValidator

Třída **TNumericValidator** definuje rozhraní a pomocné metody pro validátory ordinálních typů, jako je výčtový typ, typ byte, integer, longint apod. editované číselné podobě a to buď binárně, decimálně, hexadecimálně, příp. s pevnou řádovou čárkou. Ze třídy **TNumericValidator** je odvozeno několik speciálních tříd jako je **TByteValidator**, **TIntegerValidator** apod. (viz. hierarchie validátorů v kapitole 3.1), které doplňují několik funkcí a zjednodušují inicializaci validátorů. Proto není nutné vytvářet instance třídy **TNumericValidator**, přestože je to možné.

```
PNumericValidator = ^TNumericValidator;
TNumericValidator = object( TOrdinalValidator )
public
  FmtFlags : Byte;
  Digits   : Byte;
  Scale    : Integer;

  constructor Init( ADataPtr: Pointer; ADataFlags, AFmtFlags: Byte;
                ADigits: Byte; AScale: Integer );
  function  CharFilter( Ch: Char ): Boolean; virtual;
  function  TransferText( AText: PString; AMaxLen: Integer;
              AMode: Integer ): Integer; virtual;
  function  GetMaxTextLength: Integer; virtual;
  procedure GetAsText( AText: PString; AMaxLen: Integer;
              AValue: Longint );
```

```
    function  SetAsText( AText: PString;
                var AValue: Longint ): Integer;
    procedure GetErrorStr( AErrCode: Integer;
                var AErrStr: string ); virtual;
  end;
```

4.3.3.1. Položka TNumericValidator.FmtFlags

Položka **FmtFlags** obsahuje kombinaci příznaků nvf_ (viz. kapitola 4.1.4) určující chování validátoru. Položka je inicializovaná parametrem konstruktoru a je určena pouze pro čtení.

```
FmtFlags : Byte;
```

4.3.3.2. Položka TNumericValidator.Digits

Položka **Digits** obsahuje počet vypisovaných desetinných míst validované hodnoty převedené na znakový řetězec. Položka se uplatní pouze tehdy, jestliže v položce Flags je nastaven příznak nvfFixedPoint, tj. hodnota je zobrazována jako číslo s pevnou řádovou čárkou. Položka je inicializovaná parametrem konstruktoru a je určena pouze pro čtení.

```
Digits   : Byte;
```

4.3.3.3. Položka TNumericValidator.Scale

Položka **Scale** obsahuje měřítko, tj. hodnotu kterou je obsah validované proměnné podělen před transformací na znakový řetězec a naopak vynásoben výsledek před uložením. Položka se uplatní pouze tehdy, jestliže v položce Flags je nastaven příznak nvfFixedPoint, tj. hodnota je zobrazována jako číslo s pevnou řádovou čárkou. Položka je inicializovaná parametrem konstruktoru a je určena pouze pro čtení.

```
Scale    : Integer;
```

4.3.3.4. Konstruktor TNumericValidator.Init

Konstruktor **Init** provádí inicializaci instance třídy.

```
constructor Init( ADataPtr: Pointer; ADataFlags, AFmtFlags: Byte;
  ADigits: Byte; AScale: Integer );
```

Parametry:

ADataPtr	Ukazatel na proměnnou s validovanou hodnotu, tj. proměnnou typu **string**.
ADataFlags	Parametr obsahující kombinaci příznaků ovf_ (viz. kapitola 4.1.6) určují datový typ validované proměnné. Tento parametr je předán konstruktoru **Init** třídy **TOrdinalValidator** (viz. kapitola 4.3.2.2).
AFmtFlags	Parametr obsahující kombinaci příznaků nvf_ (viz. kapitola 4.1.4) určují formát pro editaci validované hodnoty. Jako parametr lze předat kombinaci jednoho s příznaků nvfRadixXXX (příp. nvfFixedPoint) a příznaku nvfCharFilter:

nvfRadixBin	Editace čísla v binárním podobě
nvfRadixDec	Editace čísla v decimální podobě
nvfRadixHex	Editace čísla v hexadecimální podobě
nvfFixedPoint	Editace čísla v pevné řádové čárce.
nvfCharFilter	Filtrování neplatných znaků (komponenta **TEdit**)

ADigits	Počet vypisovaných desetinných míst validované hodnoty převedené na znakový řetězec. Pokud je tento parametr nulový je hodnota před převedením zaokrouhlena na celé číslo. Parametr se uplatní pouze tehdy, jestliže parametr AFmtFlags obsahuje příznak nvfFixedPoint.
AScale	Měřítko, tj. hodnota kterou je obsah validované proměnné podělen před transformací na znakový řetězec a naopak vynásoben výsledek před uložením. Parametr se uplatní pouze tehdy, jestliže parametr AFmtFlags obsahuje příznak nvfFixedPoint.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

4.3.3.5. Metoda TNumericValidator.CharFilter

```
function  CharFilter( Ch: Char ): Boolean; virtual;
```

4.3.3.6. Metoda TNumericValidator.TransferText

```
function  TransferText( AText: PString; AMaxLen: Integer;
  AMode: Integer ): Integer; virtual;
```

4.3.3.7. Metoda TNumericValidator.GetAsText

Metoda **GetAsText** převádí hodnotu na znakový řetězec podle aktuálního nastavení položky FmtFlags (případně položek Scale a Digits).

```
procedure GetAsText( AText: PString; AMaxLen: Integer;
  AValue: Longint );
```

Parametry:

AText	Ukazatel na buffer do kterého bude uložena textová podoba parametru AValue. Maximální délka tohoto bufferu je uvedena v položce AMaxLen.
AMaxLen	Parametr AMaxLen obsahuje maximální délku řetězce, který lze uložit do bufferu specifikovaného parametrem AText.
AValue	Převáděná hodnota.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Pokud buffer AText není dostatečně velký pro textovou reprezentaci parametru AValue, pak je do něj uložen jeden znak '#'.

Metodu **GetAsText** využívá interně metoda **TransferText** (viz. kapitola 4.3.3.6).

4.3.3.8. Metoda TNumericValidator.SetAsText

Metoda **SetAsText** převádí znakový řetězec na hodnotu podle aktuálního nastavení položky FmtFlags (případně položky Scale).

```
function  SetAsText( AText: PString; var AValue: Longint ): Integer;
```

Parametry:

AText	Ukazatel na proměnnou text, který bude metodou převáděn na hodnotu.
AValue	Odkaz na proměnnou, do které bude uložen výsledek operace převodu.

Návratové hodnoty:

V případě, že převod proběhl úspěšně, vrací metoda konstanty veOk. V opačném případě, došlo k syntaktické chybě při převodu, metoda vrací konstantu veSyntax.

Poznámky:

Metodu **SetAsText** využívá interně metoda **TransferText** (viz. kapitola 4.3.3.6).

4.3.3.9. Metoda TNumericValidator.GetErrorStr

Metoda **GetErrorStr** převádí chybový kód ve_ (viz. kapitola 4.1.2] na textový řetězec.

```
procedure GetErrorStr( AErrCode: Integer;
  var AErrStr: string ); virtual;
```

Parametry:

AErrCode	Chybový kód ve_ (viz. kapitola 4.1.2).
AErrStr	Buffer, do kterého bude uložen text popisující chybu danou parametrem AErrCode.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

!@# zatím není dobře implementováno == dodělat a dopsat popis

4.3.4. Třída TByteValidator

Třída **TByteValidator** definuje validátor datového typu Byte. Obsahuje dva konstruktory **Init** a **InitFxp**. Základní konstruktor **Init** (viz. kapitola 4.3.4.3) inicializuje třídu pro validaci celého čísla s nastavitelným rozsahem v binárním, decimálním nebo hexadecimálním tvaru. Konstruktor **InitFxp** (viz. kapitola 4.3.4.4) inicializuje třídu pro validaci čísla s nastavitelnou pevnou řádovou čárkou, rozsahem a počtem desetinných míst.

```
PByteValidator = ^TByteValidator;
TByteValidator = object( TNumericValidator )
public
  Min : Byte;
  Max : Byte;

  constructor Init( AData: PByte; AFmtFlags: Byte;
                AMin, AMax: Byte );
  constructor InitFxp( AData: PByte; AFmtFlags: Byte;
                AMin, AMax: Byte; ADigits: Byte; AScale: Integer );

  procedure  GetMinMax( var AMin, AMax: Longint ); virtual;
end;
```

4.3.4.1. Položka TByteValidator.Min

Položka **Min** určuje dolní mez validované hodnoty. Položka je inicializovaná konstruktorem instance a je určena pouze pro čtení.

```
Min : Byte;
```

4.3.4.2. Položka TByteValidator.Max

Položka **Max** určuje horní mez validované hodnoty. Položka je inicializovaná konstruktorem instance a je určena pouze pro čtení.

```
Max : Byte;
```

4.3.4.3. Konstruktor TByteValidator.Init

Konstruktor **Init** inicializuje třídu pro validaci celého čísla s nastavitelném rozsahu v binárním, decimálním nebo hexadecimálním tvaru.

```
constructor Init( AData: PByte; AFmtFlags: Byte; AMin, AMax: Byte );
```

Parametry:

AData Odkaz na proměnnou s validovanou hodnotou, tj. proměnnou

typu Byte.

AFmtFlags — Parametr obsahující kombinaci příznaků nvf_ (viz. kapitola 4.1.4) určují formát pro editaci validované hodnoty. Jako parametr lze předat kombinaci jednoho s příznaků nvfRadixXXX a příznaku nvfCharFilter:

nvfRadixBin	Editace čísla v binárním podobě
nvfRadixDec	Editace čísla v decimální podobě
nvfRadixHex	Editace čísla v hexadecimální podobě
nvfCharFilter	Filtrování neplatných znaků (komponenta **TEdit**)

AMin — Dolní mez rozsahu pro validaci hodnoty.
AMax — Horní mez rozsahu pro validaci hodnoty.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Konstruktor Init nastaví navíc položku Scale na hodnotu 1 a položku Digits na hodnotu 0.

4.3.4.4. Konstruktor TByteValidator.InitFxp

Konstruktor **InitFxp** inicializuje třídu pro validaci čísla s nastavitelnou pevnou řádovou čárkou, rozsahem a počtem desetinných míst.

```
constructor InitFxp( AData: PByte; AFmtFlags: Byte; AMin, AMax: Byte;
               ADigits: Byte; AScale: Integer );
```

Parametry:

AData — Odkaz na proměnnou s validovanou hodnotou, tj. proměnnou typu Byte.
AFmtFlags — Parametr obsahující kombinaci příznaků nvf_ (viz. kapitola 4.1.4) určují formát pro editaci validované hodnoty. Jako parametr lze předat buď 0 nebo konstantu nvfCharFilter.
AMin — Dolní mez rozsahu pro validaci hodnoty.
AMax — Horní mez rozsahu pro validaci hodnoty.
ADigits — Počet zobrazených desetinných míst.
AScale — Měřítko, tj. číslo jímž budě před hodnota zobrazením podělena, a před uložením zpět do proměnně vynásobena.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Konstruktor **InitFxp** volá konstruktor předka, tj. třídy **TNumericValidator** s příznakem nvfFixedPoint. Příznaky nvfRadixXXX jsou konstruktorem ignorovány.

4.3.4.5. Metoda TByteValidator.GetMinMax

Metoda **GetMinMax** zjišťuje platný rozsah validované hodnoty.

```
procedure GetMinMax( var AMin, AMax: Longint ); virtual;
```

Parametry:

AMin	Odkaz na proměnnou, do které bude uložena minimální hodnota.
AMax	Odkaz na proměnnou, do které bude uložena maximální hodnota.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Metoda nastavuje parametry AMin a Max na hodnoty položek Min a Max.

4.3.5. Třída TShortintValidator

Třída **TShortValidator** definuje validátor datového typu ShortInt. Obsahuje dva konstruktory **Init** a **InitFxp**. Základní konstruktor **Init** (viz. kapitola 4.3.4.3) inicializuje třídu pro validaci celého čísla s nastavitelným rozsahem v binárním, decimálním nebo hexadecimálním tvaru. Konstruktor **InitFxp** (viz. kapitola 4.3.4.4) inicializuje třídu pro validaci čísla s nastavitelnou pevnou řádovou čárkou, rozsahem a počtem desetinných míst.

```
PShortIntValidator = ^TShortIntValidator;
TShortIntValidator = object( TNumericValidator )
public
  Min : ShortInt;   { Minimalni hodnota }
  Max : ShortInt;   { Maximalni hodnota }

  constructor Init( AData: PShortInt; AFmtFlags: Byte;
                AMin, AMax: ShortInt );
  constructor InitFxp( AData: PShortInt; AFmtFlags: Byte;
                AMin, AMax: ShortInt;
                ADigits: Byte; AScale: Integer );

  procedure  GetMinMax( var AMin, AMax: Longint ); virtual;
end;
```

4.3.5.1. Položka TShortintValidator.Min

Položka **Min** určuje dolní mez validované hodnoty. Položka je inicializovaná konstruktorem instance a je určena pouze pro čtení.

```
Min : ShortInt;
```

4.3.5.2. Položka TShortintValidator.Max

Položka **Max** určuje horní mez validované hodnoty. Položka je inicializovaná konstruktorem instance a je určena pouze pro čtení.

```
Max : ShortInt;
```

4.3.5.3. Konstruktor TShortintValidator.Init

Konstruktor **Init** inicializuje třídu pro validaci celého čísla s nastavitelném rozsahu v binárním, decimálním nebo hexadecimálním tvaru.

```
constructor Init( AData: PShortInt; AFmtFlags: Byte;
  AMin, AMax: ShortInt );
```

Parametry:

AData	Odkaz na proměnnou s validovanou hodnotou, tj. proměnnou typu ShortInt.
AFmtFlags	Parametr obsahující kombinaci příznaků nvf_ (viz. kapitola 4.1.4) určují formát pro editaci validované hodnoty. Jako parametr lze předat kombinaci jednoho s příznaků nvfRadixXXX a příznaku nvfCharFilter: nvfRadixBin Editace čísla v binárním podobě nvfRadixDec Editace čísla v decimální podobě nvfRadixHex Editace čísla v hexadecimální podobě nvfCharFilter Filtrování neplatných znaků (komponenta **TEdit**)
AMin	Dolní mez rozsahu pro validaci hodnoty.
AMax	Horní mez rozsahu pro validaci hodnoty.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Konstruktor Init nastaví navíc položku Scale na hodnotu 1 a položku Digits na hodnotu 0.

4.3.5.4. Konstruktor TShortintValidator.InitFxp

Konstruktor **InitFxp** inicializuje třídu pro validaci čísla s nastavitelnou pevnou řádovou čárkou, rozsahem a počtem desetinných míst.

```
constructor InitFxp( AData: PShortInt; AFmtFlags: Byte;
  AMin, AMax: ShortInt; ADigits: Byte; AScale: Integer );
```

Parametry:

AData	Odkaz na proměnnou s validovanou hodnotou, tj. proměnnou typu ShortInt.
AFmtFlags	Parametr obsahující kombinaci příznaků nvf_ (viz. kapitola 4.1.4) určují formát pro editaci validované hodnoty. Jako parametr lze předat buď 0 nebo konstantu nvfCharFilter.
AMin	Dolní mez rozsahu pro validaci hodnoty.
AMax	Horní mez rozsahu pro validaci hodnoty.
ADigits	Počet zobrazených desetinných míst.
AScale	Měřítko, tj. číslo jímž budě před hodnota zobrazením podělena, a před uložením zpět do proměnně vynásobena.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Konstruktor **InitFxp** volá konstruktor předka, tj. třídy **TNumericValidator** s příznakem nvfFixedPoint. Příznaky nvfRadixXXX jsou konstruktorem ignorovány.

4.3.5.5. Metoda TShortintValidator.GetMinMax

Metoda **GetMinMax** zjišťuje platný rozsah validované hodnoty.

```
procedure GetMinMax( var AMin, AMax: Longint ); virtual;
```

Parametry:

AMin	Odkaz na proměnnou, do které bude uložena minimální hodnota.
AMax	Odkaz na proměnnou, do které bude uložena maximální hodnota.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Metoda nastavuje parametry AMin a Max na hodnoty položek Min a Max.

4.3.6. Třída TWordValidator

Třída **TWordValidator** definuje validátor datového typu Word. Obsahuje dva konstruktory **Init** a **InitFxp**. Základní konstruktor **Init** (viz. kapitola 4.3.4.3) inicializuje třídu pro validaci celého čísla s nastavitelným rozsahem v binárním, decimálním nebo hexadecimálním tvaru. Konstruktor **InitFxp** (viz. kapitola 4.3.4.4) inicializuje třídu pro validaci čísla s nastavitelnou pevnou řádovou čárkou, rozsahem a

počtem desetinných míst.

```
PWordValidator = ^TWordValidator;
TWordValidator = object( TNumericValidator )
public
  Min : Word;       { Minimalni hodnota }
  Max : Word;       { Maximalni hodnota }

  constructor Init( AData: PWord; AFmtFlags: Byte;
                AMin, AMax: Word );
  constructor InitFxp( AData: PWord; AMin, AMax: Word;
                ADigits: Byte; AScale: Integer );
  procedure  GetMinMax( var AMin, AMax: Longint ); virtual;
end;
```

4.3.6.1. Položka TWordValidator.Min

Položka **Min** určuje dolní mez validované hodnoty. Položka je inicializovaná konstruktorem instance a je určena pouze pro čtení.

```
Min : Word;
```

4.3.6.2. Položka TWordValidator.Max

Položka **Max** určuje horní mez validované hodnoty. Položka je inicializovaná konstruktorem instance a je určena pouze pro čtení.

```
Max : Word;
```

4.3.6.3. Konstruktor TWordValidator.Init

Konstruktor **Init** inicializuje třídu pro validaci celého čísla s nastavitelném rozsahu v binárním, decimálním nebo hexadecimálním tvaru.

```
constructor Init( AData: PWord; AFmtFlags: Byte; AMin, AMax: Word );
```

Parametry:

AData	Odkaz na proměnnou s validovanou hodnotou, tj. proměnnou typu Word.
AFmtFlags	Parametr obsahující kombinaci příznaků nvf_ (viz. kapitola 4.1.4) určují formát pro editaci validované hodnoty. Jako parametr lze předat kombinaci jednoho s příznaků nvfRadixXXX a příznaku nvfCharFilter:
	nvfRadixBin Editace čísla v binárním podobě nvfRadixDec Editace čísla v decimální podobě nvfRadixHex Editace čísla v hexadecimální podobě nvfCharFilter Filtrování neplatných znaků (komponenta **TEdit**)
AMin	Dolní mez rozsahu pro validaci hodnoty.
AMax	Horní mez rozsahu pro validaci hodnoty.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Konstruktor Init nastaví navíc položku Scale na hodnotu 1 a položku Digits na hodnotu 0.

4.3.6.4. Konstruktor TWordValidator.InitFxp

Konstruktor **InitFxp** inicializuje třídu pro validaci čísla s nastavitelnou pevnou řádovou čárkou, rozsahem a počtem desetinných míst.

```
constructor InitFxp( AData: PWord; AFmtFlags: Byte;
  AMin, AMax: Word; ADigits: Byte; AScale: Integer );
```

Parametry:

AData	Odkaz na proměnnou s validovanou hodnotou, tj. proměnnou typu Word.
AFmtFlags	Parametr obsahující kombinaci příznaků nvf_ (viz. kapitola 4.1.4) určují formát pro editaci validované hodnoty. Jako parametr lze předat buď 0 nebo konstantu nvfCharFilter.
AMin	Dolní mez rozsahu pro validaci hodnoty.
AMax	Horní mez rozsahu pro validaci hodnoty.
ADigits	Počet zobrazených desetinných míst.
AScale	Měřítko, tj. číslo jímž budě před hodnota zobrazením podělena, a před uložením zpět do proměnně vynásobena.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Konstruktor **InitFxp** volá konstruktor předka, tj. třídy **TNumericValidator** s příznakem nvfFixedPoint. Příznaky nvfRadixXXX jsou konstruktorem ignorovány.

4.3.6.5. Metoda TWordValidator.GetMinMax

Metoda **GetMinMax** zjišťuje platný rozsah validované hodnoty.

```
procedure GetMinMax( var AMin, AMax: Longint ); virtual;
```

Parametry:

AMin	Odkaz na proměnnou, do které bude uložena minimální hodnota.
AMax	Odkaz na proměnnou, do které bude uložena maximální hodnota.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Metoda nastavuje parametry AMin a Max na hodnoty položek Min a Max.

4.3.7. Třída TIntegerValidator

Třída **TIntegerValidator** definuje validátor datového typu Integer. Obsahuje dva konstruktory **Init** a **InitFxp**. Základní konstruktor **Init** (viz. kapitola 4.3.4.3) inicializuje třídu pro validaci celého čísla s nastavitelným rozsahem v binárním, decimálním nebo hexadecimálním tvaru. Konstruktor **InitFxp** (viz. kapitola 4.3.4.4) inicializuje třídu pro validaci čísla s nastavitelnou pevnou řádovou čárkou, rozsahem a počtem desetinných míst.

```
PIntegerValidator = ^TIntegerValidator;
TIntegerValidator = object( TNumericValidator )
public
  Min : Integer;
  Max : Integer;

  constructor Init( AData: PInteger; AFmtFlags: Byte;
                AMin, AMax: Integer );
  constructor InitFxp( AData: PInteger; AFmtFlags: Byte;
                AMin, AMax: Integer;
                ADigits: Byte; AScale: Integer );

  procedure  GetMinMax( var AMin, AMax: Longint ); virtual;
end;
```

4.3.7.1. Položka TIntegerValidator.Min

Položka **Min** určuje dolní mez validované hodnoty. Položka je inicializovaná konstruktorem instance a je určena pouze pro čtení.

```
Min : Integer;
```

4.3.7.2. Položka TIntegerValidator.Max

Položka **Max** určuje horní mez validované hodnoty. Položka je inicializovaná konstruktorem instance a je určena pouze pro čtení.

```
Max : Integer;
```

4.3.7.3. Konstruktor TIntegerValidator.Init

Konstruktor **Init** inicializuje třídu pro validaci celého čísla s nastavitelném rozsahu v binárním, decimálním nebo hexadecimálním tvaru.

```
constructor Init( AData: PInteger; AFmtFlags: Byte;
  AMin, AMax: Integer );
```

Parametry:

AData	Odkaz na proměnnou s validovanou hodnotou, tj. proměnnou typu Integer.
AFmtFlags	Parametr obsahující kombinaci příznaků nvf_ (viz. kapitola 4.1.4) určují formát pro editaci validované hodnoty. Jako parametr lze předat kombinaci jednoho s příznaků nvfRadixXXX a příznaku nvfCharFilter: nvfRadixBin Editace čísla v binárním podobě nvfRadixDec Editace čísla v decimální podobě nvfRadixHex Editace čísla v hexadecimální podobě nvfCharFilter Filtrování neplatných znaků (komponenta **TEdit**)
AMin	Dolní mez rozsahu pro validaci hodnoty.
AMax	Horní mez rozsahu pro validaci hodnoty.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Konstruktor Init nastaví navíc položku Scale na hodnotu 1 a položku Digits na hodnotu 0.

4.3.7.4. Konstruktor TIntegerValidator.InitFxp

Konstruktor **InitFxp** inicializuje třídu pro validaci čísla s nastavitelnou pevnou řádovou čárkou, rozsahem a počtem desetinných míst.

```
constructor InitFxp( AData: PInteger; AFmtFlags: Byte;
  AMin, AMax: Integer; ADigits: Byte; AScale: Integer );
```

Parametry:

AData	Odkaz na proměnnou s validovanou hodnotou, tj. proměnnou typu Integer.
AFmtFlags	Parametr obsahující kombinaci příznaků nvf_ (viz. kapitola 4.1.4) určují formát pro editaci validované hodnoty. Jako parametr lze předat buď 0 nebo konstantu nvfCharFilter.
AMin	Dolní mez rozsahu pro validaci hodnoty.
AMax	Horní mez rozsahu pro validaci hodnoty.
ADigits	Počet zobrazených desetinných míst.
AScale	Měřítko, tj. číslo jímž budě před hodnota zobrazením podělena, a před uložením zpět do proměnně vynásobena.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Konstruktor **InitFxp** volá konstruktor předka, tj. třídy **TNumericValidator** s příznakem nvfFixedPoint. Příznaky nvfRadixXXX jsou konstruktorem ignorovány.

4.3.7.5. Metoda TIntegerValidator.GetMinMax

Metoda **GetMinMax** zjišťuje platný rozsah validované hodnoty.

```
procedure GetMinMax( var AMin, AMax: Longint ); virtual;
```

Parametry:

AMin	Odkaz na proměnnou, do které bude uložena minimální hodnota.
AMax	Odkaz na proměnnou, do které bude uložena maximální hodnota.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Metoda nastavuje parametry AMin a Max na hodnoty položek Min a Max.

4.3.8. Třída TLongintValidator

Třída **TLongintValidator** definuje validátor datového typu Longint. Obsahuje dva konstruktory **Init** a **InitFxp**. Základní konstruktor **Init** (viz. kapitola 4.3.4.3) inicializuje třídu pro validaci celého čísla s nastavitelným rozsahem v binárním, decimálním nebo hexadecimálním tvaru. Konstruktor **InitFxp** (viz. kapitola 4.3.4.4) inicializuje třídu pro validaci čísla s nastavitelnou pevnou řádovou čárkou, rozsahem a počtem desetinných míst.

```
PLongintValidator = ^TLongintValidator;
TLongintValidator = object( TNumericValidator )
public
  Min : Longint;
  Max : Longint;

  constructor Init( AData: PLongint; AFmtFlags: Byte;
                AMin, AMax: Longint );
  constructor InitFxp( AData: PLongint; AFmtFlags: Byte;
                AMin, AMax: Longint;
                ADigits: Byte; AScale: Integer );

  procedure GetMinMax( var AMin, AMax: Longint ); virtual;
end;
```

4.3.8.1. Položka TLongintValidator.Min

Položka **Min** určuje dolní mez validované hodnoty. Položka je inicializovaná konstruktorem instance a je určena pouze pro čtení.

```
Min : Longint;
```

4.3.8.2. Položka TLongintValidator.Max

Položka **Max** určuje horní mez validované hodnoty. Položka je inicializovaná konstruktorem instance a je určena pouze pro čtení.

```
Max : Longint;
```

4.3.8.3. Konstruktor TLongintValidator.Init

Konstruktor **Init** inicializuje třídu pro validaci celého čísla s nastavitelném rozsahu v binárním, decimálním nebo hexadecimálním tvaru.

```
constructor Init( AData: PLongint; AFmtFlags: Byte;
  AMin, AMax: Longint );
```

Parametry:

AData	Odkaz na proměnnou s validovanou hodnotou, tj. proměnnou typu Longint.
AFmtFlags	Parametr obsahující kombinaci příznaků nvf_ (viz. kapitola 4.1.4) určují formát pro editaci validované hodnoty. Jako parametr lze předat kombinaci jednoho s příznaků nvfRadixXXX a příznaku nvfCharFilter: nvfRadixBin Editace čísla v binárním podobě nvfRadixDec Editace čísla v decimální podobě nvfRadixHex Editace čísla v hexadecimální podobě nvfCharFilter Filtrování neplatných znaků (komponenta **TEdit**)
AMin	Dolní mez rozsahu pro validaci hodnoty.
AMax	Horní mez rozsahu pro validaci hodnoty.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Konstruktor Init nastaví navíc položku Scale na hodnotu 1 a položku Digits na hodnotu 0.

4.3.8.4. Konstruktor TLongintValidator.InitFxp

Konstruktor **InitFxp** inicializuje třídu pro validaci čísla s nastavitelnou pevnou

řádovou čárkou, rozsahem a počtem desetinných míst.

```
constructor InitFxp( AData: PShortInt; AFmtFlags: Byte;
  AMin, AMax: Longint; ADigits: Byte; AScale: Integer );
```

Parametry:

AData	Odkaz na proměnnou s validovanou hodnotou, tj. proměnnou typu Longint.
AFmtFlags	Parametr obsahující kombinaci příznaků nvf_ (viz. kapitola 4.1.4) určují formát pro editaci validované hodnoty. Jako parametr lze předat buď 0 nebo konstantu nvfCharFilter.
AMin	Dolní mez rozsahu pro validaci hodnoty.
AMax	Horní mez rozsahu pro validaci hodnoty.
ADigits	Počet zobrazených desetinných míst.
AScale	Měřítko, tj. číslo jímž budě před hodnota zobrazením podělena, a před uložením zpět do proměnně vynásobena.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Konstruktor **InitFxp** volá konstruktor předka, tj. třídy **TNumericValidator** s příznakem nvfFixedPoint. Příznaky nvfRadixXXX jsou konstruktorem ignorovány.

4.3.8.5. Metoda TLongintValidator.GetMinMax

Metoda **GetMinMax** zjišťuje platný rozsah validované hodnoty.

```
procedure GetMinMax( var AMin, AMax: Longint ); virtual;
```

Parametry:

AMin	Odkaz na proměnnou, do které bude uložena minimální hodnota.
AMax	Odkaz na proměnnou, do které bude uložena maximální hodnota.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Metoda nastavuje parametry AMin a Max na hodnoty položek Min a Max.

4.3.9. Třída TTextValidator

Abstraktní třída **TTextValidator** definuje bázovou třídy pro validátory znakových

datových typů. Potomci této třídy musí minimálně předefinovat metody **GetDataSize** a **TransferText**.

```
PTextValidator = ^TTextValidator;
TTextValidator = object( TValidator )
public
  FmtFlags : Byte;
  MaxLen   : Byte;

  constructor Init( ADataPtr: Pointer; AMaxLen: Byte;
                AFmtFlags: Byte );

  function  ValueFilter( const AValue: string ): Integer; virtual;
  function  GetMaxTextLength: Integer; virtual;
  procedure Convert( var AValue: string ); virtual;
end;
```

4.3.9.1. Položka TTextValidator.FmtFlags

Položka **FmtFlags** upřesňuje chování validátoru. Obsahuje kombinaci příznaků tvf_ (viz. kapitola 4.1.5). Položka je inicializována parametrem konstruktoru a je určena pouze pro čtení.

```
FmtFlags : Byte;
```

4.3.9.2. Položka TTextValidator.MaxLen

Položka **MaxLen** obsahuje maximální délku textového řetězce, tj. maximální počet znaků, které lze zapsat do datové struktury, na kterou ukazuje metoda **GetDataPtr**. Položka je inicializována parametrem konstruktoru a je určena pouze pro čtení.

```
MaxLen : Byte;
```

Položka MaxLen může být dále využita k validaci řetězce. Pokud je v položce FmtFlags nastaven příznak tvfCheckLength, kontroluje metoda **ValueFilter** délku řetězce. Pokud je větší než udává položka MaxLen, pak tato metoda vrací chybový kód veLength.

4.3.9.3. Konstruktor TTextValidator.Init

Konstruktor **Init** provádí inicializaci instance třídy.

```
constructor Init( ADataPtr: Pointer; AMaxLen: Byte;
              AFmtFlags: Byte );
```

Parametry:

ADataPtr	Ukazatel na validovaná data.
AMaxLen	Maximální délka řetězce. Tento parametr bude konstruktore
AFmtFlags	Kombinace příznaků tvf_ (viz. kapitola 4.1.5) upřesňujících chování validátoru.

Návratové hodnoty:

Konstruktor nevrací žádnou hodnotu.

Poznámky:

Parametr konstruktoru ADataPtr není ukazatelem na konkrétní datový typ. Tento typ musí upřesnit teprve potomci třídy **TTextValidator**.

4.3.9.4. Metoda TTextValidator.ValueFilter

Metoda **ValueFilter** provádí validaci textového řetězce.

```
function  ValueFilter( const AValue: string ): Integer; virtual;
```

Parametry:

AValue Validovaný řetězec.

Návratové hodnoty:

Metoda vrací hodnotu veOk, pokud je obsah znakového řetězce v pořádku. V opačném případě vrací metoda jednu z chybový konstant ve_ (viz. kapitola 4.1.2).

Poznámky:

Ve třídě **TTextValidator** je metoda **ValueFilter** implementovaná tak, že kontroluje pouze délku řetězce a to jen tehdy, jestliže položka FmtFlags obsahuje příznak tvfCheckLength. V případě, že je délka řetězce větší, než je maximální zadaná délka řetězce, vrací metoda hodnotu veLength.

Potomci třídy mohou tuto metodu předefinovat, tak aby prováděla další typy kontrol.

4.3.9.5. Metoda TTextValidator.Convert

Metoda **Convert** provádí konverzi znakového řetězce před uložením.

```
procedure Convert( var AValue: string ); virtual;
```

Parametry:

AValue Konvertovaný řetězec.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Metoda **Convert** je volána před uložením editovaného řetězce zpět na místo, na které ukazuje metoda **GetDataPtr**. Metoda **Convert** umožnuje implicitně odstranit mezery zleva nebo zprava, převést znaky řetězce na malá nebo velká písmena (viz. položka FmtFlags v kapitole 4.3.9.1)

4.3.10. Třída TStringValidator

Třída **TStringValidator** definuje třídu pro validaci proměnné typu **string**. Tato třída dědí všechny vlastnosti třídy **TTextValidator** (viz. kapitola 4.3.9)

```
PStringValidator = ^TStringValidator;
TStringValidator = object( TTextValidator )
public
  constructor Init( ADataPtr: PString; AFmtFlags: Byte;
                AMaxLen: Byte );

  function  GetDataSize: Word; virtual;
  function  TransferText( AText: PString; AMaxLen: Integer;
              AMode: Integer ): Integer; virtual;
end;
```

4.3.10.1. Konstruktor TStringValidator.Init

Konstruktor **Init** provádí inicializaci instance třídy.

```
constructor Init( ADataPtr: PString; AFmtFlags: Byte;
  AMaxLen: Byte );
```

Parametry:

ADataPtr	Ukazatel na proměnnou s validovanou hodnotu, tj. proměnnou typu **string**.
AFmtFlags	Příznaky tvf_ určující chování validátory (viz. kapitola 4.1.5)
AMaxLen	Maximální délka řetězce, na který ukazuje parametr ADataPtr.

Návratové hodnoty:

Konstruktor nevrací žádnou hodnotu.

Poznámky:

4.3.10.2. Metoda TStringValidator.GetDataSize

Metoda **GetDataSize** vrací délku binárních dat na které ukazuje ukazatel vrácený metodou **GetDataPtr**.

```
function  GetDataSize: Word; virtual;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda vrací délku dat v bajtech.

Poznámky:

Metoda **TStringValidator.GetDataSize** předefinována abstraktní metodu **TValidator.GetDataSize** a vrací hodnotu položky MaxLen zvýšenou o jedna.

4.3.10.3. Metoda TStringValidator.TransferText

Metoda **TransferText** provádí převod binární hodnoty svázané s validátorem na textový řetězec a naopak, dále provádí validaci tohoto textového řetězce.

```
function  TransferText( AText: PString; AMaxLen: Integer;
            AMode: Integer ): Integer; virtual;
```

Parametry:

AText	Ukazatel na buffer pro textový řetězec. Délka bufferu je dána parametrem AMaxLen. Tento buffer je určen buď pro zápis nebo pro čtení, podle nastavení parametru AMode.
AMaxLen	Délka bufferu pro textový řetězec, parametr AMaxLen je brán v potaz pouze tehdy, jestliže parametr AMode je nastaven na hodnotu vmLoad. V opačném případě může být hodnota tohoto parametru libovolná.
AMode	Určuje chování metody a může obsahovat tři různé hodnoty:

vmLoad	Metoda provede konverzi binární hodnoty, na kterou ukazuje funkce **GetDataPtr** na textový řetězec.
vmStore	Metoda provede validaci a konverzi textového řetězce na binární hodnotu, na kterou ukazuje funkce **GetDataPtr**.
vmValidate	Metoda provede pouze validaci textového řetězce.

Návratové hodnoty:

Metoda **TransferText** vrací jednu z konstant s prefixem ve_ (viz. kapitola 4.1.2). V případě úspěšného provedení vrací konstantu veOk.

Poznámky:

Metoda **TStringValidator** provádí validaci hodnoty pomocí metody **ValueFilter** (viz. kapitola 4.3.9.4). Před uložení hodnoty do proměnné, na kterou ukazuje metoda **GetDataPtr** je zavolána metoda **Convert** (viz. kapitola 4.3.9.5)

4.3.11. Třída TCharArrayValidator

Třída **TCharArrayValidator** definuje třídu pro validaci proměnné typu pole znaků, tedy **array** […] **of** Char. Tato třída dědí všechny vlastnosti třídy **TTextValidator** (viz. kapitola 4.3.9)

```
  PCharArrayValidator = ^TCharArrayValidator;
  TCharArrayValidator = object( TTextValidator )
```

```
  public
    FillChar : Char;

    constructor Init( ADataPtr: PChar; AFmtFlags: Byte;
                  AMaxLen: Byte; AFillChar: Char );

    function  GetDataSize: Word; virtual;
    function  TransferText( AText: PString; AMaxLen: Integer;
                AMode: Integer ): Integer; virtual;
  end;
```

4.3.11.1. Položka TCharArrayValidator.FillChar

Položka **FillChar** určuje znak, kterým bude vyplněna nevyužitá část pole znaků. Obvykle je nastavena na hodnotu #0 nebo #32 (znak mezera). Položka je inicializována parametrem konstruktoru a je určena pouze pro čtení.

```
FillChar : Char;
```

4.3.11.2. Konstruktor TCharArrayValidator.Init

Konstruktor **Init** provádí inicializaci instance třídy.

```
constructor Init( ADataPtr: PChar; AFmtFlags: Byte;
              AMaxLen: Byte; AFillChar: Char );
```

Parametry:

ADataPtr	Ukazatel na proměnnou s validovanou hodnotou, tj. proměnnou pole typu **array** […] **of** Char. Délku tohoto pole udává parametr AMaxLen.
AFmtFlags	Příznaky tvf_ určující chování validátory (viz. kapitola 4.1.5)
AMaxLen	Délka pole znaků, na který ukazuje parametr ADataPtr.
AFillChar	Znak, kterým bude vyplněna nevyužitá část pole znaků.

Návratové hodnoty:

Konstruktor nevrací žádnou hodnotu.

Poznámky:

4.3.11.3. Metoda TCharArrayValidator.GetDataSize

Metoda **GetDataSize** vrací délku binárních dat na které ukazuje ukazatel vrácený metodou **GetDataPtr**.

```
function  GetDataSize: Word; virtual;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda vrací délku dat v bajtech.

Poznámky:

Metoda **TCharArrayValidator.GetDataSize** předefinována abstraktní metodu **TValidator.GetDataSize** a vrací hodnotu položky MaxLen.

4.3.11.4. Metoda TCharArrayValidator.TransferText

Metoda **TransferText** provádí převod binární hodnoty svázané s validátorem na textový řetězec a naopak, dále provádí validaci tohoto textového řetězce.

```
function  TransferText( AText: PString; AMaxLen: Integer;
            AMode: Integer ): Integer; virtual;
```

Parametry:

AText	Ukazatel na buffer pro textový řetězec. Délka bufferu je dána parametrem AMaxLen. Tento buffer je určen buď pro zápis nebo pro čtení, podle nastavení parametru AMode.
AMaxLen	Délka bufferu pro textový řetězec, parametr AMaxLen je brán v potaz pouze tehdy, jestliže parametr AMode je nastaven na hodnotu vmLoad. V opačném případě může být hodnota tohoto parametru libovolná.
AMode	Určuje chování metody a může obsahovat tři různé hodnoty:

vmLoad	Metoda provede konverzi binární hodnoty, na kterou ukazuje funkce **GetDataPtr** na textový řetězec.
vmStore	Metoda provede validaci a konverzi textového řetězce na binární hodnotu, na kterou ukazuje funkce **GetDataPtr**.
vmValidate	Metoda provede pouze validaci textového řetězce.

Návratové hodnoty:

Metoda **TransferText** vrací jednu z konstant s prefixem ve_ (viz. kapitola 4.1.2). V případě úspěšného provedení vrací konstantu veOk.

Poznámky:

Metoda **TCharArrayValidator** provádí validaci hodnoty pomocí metody **ValueFilter** (viz. kapitola 4.3.9.4). Před uložení hodnoty do proměnné, na kterou ukazuje metoda **GetDataPtr** je zavolána metoda **Convert** (viz. kapitola 4.3.9.5)

4.3.12. Třída TRealValidator

Třída **TRealValidator** definuje validátor datového typu Real.

```
PRealValidator = ^TRealValidator;
TRealValidator = object( TValidator )
public
  FmtFlags : Byte;
  Min      : Real;
  Max      : Real;
  Scale    : Real;
  Digits   : Byte;

  constructor Init( ADataPtr: PReal; AFmtFlags: Byte;
              AMin, AMax: Real; ADigits: Byte; AScale: Real );

  function  GetDataSize: Word; virtual;
  function  ValueFilter( AValue: Real ): Integer; virtual;
  function  TransferText( AText: PString; AMaxLen: Integer;
            AMode: Integer ): Integer; virtual;
  function  GetMaxTextLength: Integer; virtual;
  function  CharFilter( Ch: Char ): Boolean; virtual;
end;
```

4.3.12.1. Položka TRealValidator.FmtFlags

Položka **FmtFlags** obsahuje kombinaci příznaků rvf_ (viz. kapitola 4.1.6) určující chování validátoru. Položka je inicializovaná parametrem konstruktoru a je určena pouze pro čtení.

```
FmtFlags : Byte;
```

4.3.12.2. Položka TRealValidator.Min

Položka **Min** určuje dolní mez validované hodnoty. Položka je inicializovaná konstruktorem instance a je určena pouze pro čtení.

```
Min : Real;
```

4.3.12.3. Položka TRealValidator.Max

Položka **Max** určuje horní mez validované hodnoty. Položka je inicializovaná konstruktorem instance a je určena pouze pro čtení.

```
Max : Real;
```

4.3.12.4. Položka TRealValidator.Scale

Položka **Scale** obsahuje měřítko, tj. hodnotu kterou je obsah validované proměnné podělen před transformací na znakový řetězec a naopak vynásoben výsledek před uložením. Položka je inicializovaná parametrem konstruktoru a je určena pouze pro čtení.

```
Scale    : Real;
```

4.3.12.5. Položka TRealValidator.Digits

Položka **Digits** obsahuje počet vypisovaných desetinných míst validované hodnoty převedené na znakový řetězec. Položka je inicializovaná parametrem konstruktoru a je určena pouze pro čtení.

```
Digits   : Byte;
```

4.3.12.6. Konstruktor TRealValidator.Init

Konstruktor **Init** provádí inicializaci instance třídy.

```
constructor Init( ADataPtr: PReal; AFmtFlags: Byte;
  AMin, AMax: Real; ADigits: Byte; AScale: Real );
```

Parametry:

ADataPtr	Ukazatel na proměnnou s validovanou hodnotu, tj. proměnnou typu **string**.
AFmtFlags	Parametr obsahující kombinaci příznaků rvf_ (viz. kapitola 4.1.6) určují formát pro editaci validované hodnoty. rvfCharFilter Filtrování neplatných znaků (komponenta **TEdit**)
AMin	Dolní mez rozsahu pro validaci hodnoty.
AMax	Horní mez rozsahu pro validaci hodnoty.
ADigits	Počet vypisovaných desetinných míst validované hodnoty převedené na znakový řetězec. Pokud je tento parametr nulový je hodnota před převedením zaokrouhlena na celé číslo.
AScale	Měřítko, tj. hodnota kterou je obsah validované proměnné podělen před transformací na znakový řetězec a naopak vynásoben výsledek před uložením.

Návratové hodnoty:

Konstruktor nevrací žádnou hodnotu.

Poznámky:

Parametr AScale, tj. měřítko lze využít např. k editaci parametru v jiných jednotkách apod. Např. pokud je editovaná hodnota v [g], pak s měřítkem 1000, bude hodnota editovaná v [kg].

4.3.12.7. Metoda TRealValidator.ValueFilter

Metoda **ValueFilter** slouží k validaci hodnoty, určuje zda je zadaná hodnota platná či nikoli.

```
function  ValueFilter( AValue: Real ): Integer; virtual;
```

Parametry:

AValue	Kontrolovaná hodnota.

Návratové hodnoty:

Metoda **ValueFilter** vrací chybový kód ve_ (viz. kapitola 4.1.2). V případě, že je

zadaná hodnota v pořádku, vrací konstantu veOk.

Poznámky:

Metoda **ValueFilter** je volána v rámci metody **Transfer** s parametrem vmStore a vmValidate.

Třída **TRealValidator** implementuje metodu **ValueFilter** tak, že kontroluje pouze zda je hodnota v rozsahu, určeném položkami Min a Max.

4.3.12.8. Metoda TRealValidator.TransferText

Metoda **TransferText** provádí převod binární hodnoty svázané s validátorem na textový řetězec a naopak, dále provádí validaci tohoto textového řetězce.

```
function  TransferText( AText: PString; AMaxLen: Integer;
            AMode: Integer ): Integer; virtual;
```

Parametry:

AText	Ukazatel na buffer pro textový řetězec. Délka bufferu je dána parametrem AMaxLen. Tento buffer je určen buď pro zápis nebo pro čtení, podle nastavení parametru AMode.
AMaxLen	Délka bufferu pro textový řetězec, parametr AMaxLen je brán v potaz pouze tehdy, jestliže parametr AMode je nastaven na hodnotu vmLoad. V opačném případě může být hodnota tohoto parametru libovolná.
AMode	Určuje chování metody a může obsahovat tři různé hodnoty:

vmLoad	Metoda provede konverzi binární hodnoty, na kterou ukazuje funkce **GetDataPtr** na textový řetězec.
vmStore	Metoda provede validaci a konverzi textového řetězce na binární hodnotu, na kterou ukazuje funkce **GetDataPtr**.
vmValidate	Metoda provede pouze validaci textového řetězce.

Návratové hodnoty:

Metoda **TransferText** vrací jednu z konstant s prefixem ve_ (viz. kapitola 4.1.2). V případě úspěšného provedení vrací konstantu veOk.

Poznámky:

4.3.12.9. Metoda TRealValidator.CharFilter

Metoda **CharFilter** určuje zda zadaný znak je platný v textové reprezentaci

validovaných dat.

```
function  CharFilter( Ch: Char ): Boolean; virtual;
```

Parametry:

Ch Posuzovaný znak.

Návratové hodnoty:

Metoda vrací hodnotu True, pokud je zadaný znak platný. V opačném případě vrací hodnotu False.

Poznámky:

Pokud je nastaven příznak rvfCharFilter v položce FmtFlags vrací metoda **CharFilter** třídy **TRealValidator** hodnotu True pro znaky z množiny [‘0’..’9’, ‘E’, ‘e’, ‘-‘, ‘.’]. Pokud je tento příznak vynulován vrací metoda hodnotu True pro všechny znaky.

Metodu **CharFilter** využívá např. komponenta **TEdit**, která povolí zadat pouze platné znaky. U ostatních komponent se metoda **CharFilter** většinou nevyužívá.

4.3.13. Třída TEnumValidator

Třída **TEnumValidator** definuje obecnný validátor pro výčtové typy.

```
PEnumValidator = ^TEnumValidator;
TEnumValidator = object( TOrdinalValidator )
public
  EnumCount  : Integer;
  LookupFunc : PStringPtrArray;

  constructor Init( ADataPtr: Pointer; ADataFlags: Byte;
    AEnumCount: Integer; AEnumText: PStringPtrArray );
  function  TransferText( AText: PString; AMaxLen: Integer;
              AMode: Integer ): Integer; virtual;
  procedure GetMinMax( var AMin, AMax: Longint ); virtual;
  function EnumLookup( AValue: Longint ): PString; virtual;
end;
```

4.3.13.1. Položka TEnumValidator.EnumCount

Položka **EnumCount** obsahuje počet variant výčtového typu. Předpokládá se, že ordinální hodnoty pro jednotlivé varianty začínají hodnotou 0 a končí hodnotou EnumCount – 1. Položka EnumCount zároveň určuje délku pole znakových řetězců, na které ukazuje položka EnumText. Položka je inicializována parametrem konstruktoru a je určena pouze pro čtení.

```
EnumCount : Integer;
```

4.3.13.2. Položka TEnumValidator.LookupFunc

Položka **LookupFunc** obsahuje odkaz na funkci pro převod ordinální hodnoty

výčtového typu na text. Položka je inicializována parametrem konstruktoru a je určena pouze pro čtení.

```
LookupFunc : TEnumLookupFunc;
```

4.3.13.3. Konstruktor TEnumValidator.Init

Konstruktor **Init** provádí inicializaci instance třídy.

```
constructor Init( ADataPtr: Pointer; ADataFlags: Byte;
  AEnumCount: Integer; AEnumText: PStringPtrArray );
```

Parametry:

ADataPtr	Ukazatel na validovanou proměnnou. Datový typ této proměnné určuje parametr ADataFlags.
ADataFlags	Vlastnosti validovaných dat, resp. jejich datový typ (viz. příznaky ovf_ v kapitole 4.1.3).
AEnumCount	Počet variant výčtového typu.
ALookupFunc	Odkaz na funkci vracející převádějící ordinální hodnotu na ukazatel na znakový řetězec. Viz. poznámky:

Návratové hodnoty:

Konstruktor nevrací žádnou hodnotu.

Poznámky:

Příklad:

```
type
   TAnimal = ( anDog, anCat, anHedgehog, anMouse );

function AnimalToString( Animal: Integer ): PString;
const
   Lookup : array[0..3] of string[8] =
    ( 'Dog', ' Cat', 'Hedgehog', 'Mouse' );
begin
  AnimalToString := @Lookup[ Animal ];
end;

var
   Animal : TAnimal;

{ Inicializace validatoru }

Validator := New( PEnumValidator, Init( @Animal, ovfSizeByte,
                     High(TAnimal) + 1, AnimalToString ));
```

4.3.13.4. Metoda TEnumValidator.GetMinMax

Metoda **GetMinMax** zjišťuje platný rozsah validované hodnoty.

```
procedure GetMinMax( var AMin, AMax: Longint ); virtual;
```

Parametry:

AMin	Odkaz na proměnnou, do které bude uložena minimální hodnota.
AMax	Odkaz na proměnnou, do které bude uložena maximální hodnota.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Třída **TEnumValidator** implementuje metodu **GetMinMax** tak, že nastaví parametr AMin vždy na hodnotu 0, a parametr AMax na hodnotu položky EnumCount –1.

4.3.13.5. Metoda TEnumValidator.EnumLookup

Metoda **EnumLookup** vrací ukazatel na název varianty výčtového typu.

```
function  EnumLookup( AValue: Longint ): PString; virtual;
```

Parametry:

AValue	Ordinální hodnota příslušné varianty výčtového typu.

Návratové hodnoty:

Metoda vrací ukazatel na znakový řetězec pro příslušnou variantu výčtového typu. Pokud je parametr AValue mimo rozsah výčtového typu, nebo variantě odpovídá prázdný řetězec, metoda vrací hodnotu **nil**.

Poznámky:

Třída **TEnumValidator** předefinována metodu **EnumLookup** předka, tak že volá funkci **LookupFunc** (viz. kapitola 4.3.13.2) a vrací její návratovou hodnotu.

4.3.13.6. Metoda TEnumValidator.TransferText

Metoda **TransferText** tvoří jádro validátoru. Provádí převod binární hodnoty svázané s validátorem na textový řetězec a naopak, dále provádí validaci tohoto textového řetězce.

```
function  TransferText( AText: PString; AMaxLen: Integer;
            AMode: Integer ): Integer; virtual;
```

Parametry:

AText	Ukazatel na buffer pro textový řetězec. Délka bufferu je dána parametrem AMaxLen. Tento buffer je určen buď pro zápis nebo pro čtení, podle nastavení parametru AMode.

AMaxLen	Délka bufferu pro textový řetězec, parametr AMaxLen je brán v potaz pouze tehdy, jestliže parametr AMode je nastaven na hodnotu vmLoad. V opačném případě může být hodnota tohoto parametru libovolná.
AMode	Určuje chování metody a může obsahovat tři různé hodnoty:

vmLoad	Metoda provede konverzi binární hodnoty, na kterou ukazuje funkce **GetDataPtr** na textový řetězec.
vmStore	Metoda provede validaci a konverzi textového řetězce na binární hodnotu, na kterou ukazuje funkce **GetDataPtr**.
vmValidate	Metoda provede pouze validaci textového řetězce.

Návratové hodnoty:

Metoda **TransferText** vrací jednu z konstant s prefixem ve_ (viz. kapitola 4.1.2). V případě úspěšného provedení vrací konstantu veOk.

Poznámky:

Třída **TEnumValidator** předefinována metodu **TransferText** tak, že pro každý parametr AMode různý od vmLoad vrací chybový kód veNotImpl. Metoda provádí pouze převod binární hodnoty na znakový řetězec, pomocí metodu **EnumLookup**.

4.3.14. Třída TByteEnumValidator

Třída **TByteEnumValidator** definuje validátor pro výčtové typy nad základním datovým typem Byte.

```
PByteEnumValidator = ^TByteEnumValidator;
TByteEnumValidator = object( TEnumValidator )
public
  constructor Init( ADataPtr: PByte;
    AEnumCount: Integer; ALookupFunc: TEnumLookupFunc );
end;
```

4.3.14.1. Konstruktor TByteEnumValidator.Init

Konstruktor **Init** provádí inicializaci instance třídy.

```
constructor Init( ADataPtr: PByte; AEnumCount: Integer;
  ALookupFunc: TEnumLookupFunc );
```

Parametry:

ADataPtr	Ukazatel na validovanou proměnnou, tj. proměnnou kompatibilní s typem Byte.
AEnumCount	Počet variant výčtového typu.

ALookupFunc	Odkaz na funkci vracející převádějící ordinální hodnotu na ukazatel na znakový řetězec. Viz. poznámky:

Návratové hodnoty:

Konstruktor nevrací žádnou hodnotu.

Poznámky:

4.3.15. Třída TWordEnumValidator

Třída **TWordEnumValidator** definuje validátor pro výčtové typy nad základním datovým typem Word.

```
PWordEnumValidator = ^TWordEnumValidator;
TWordEnumValidator = object( TEnumValidator )
public
  constructor Init( ADataPtr: PWord;
    AEnumCount: Integer; ALookupFunc: TEnumLookupFunc );
end;
```

4.3.15.1. Konstruktor TWordEnumValidator.Init

Konstruktor **Init** provádí inicializaci instance třídy.

```
constructor Init( ADataPtr: PWord; AEnumCount: Integer;
  ALookupFunc: TEnumLookupFunc );
```

Parametry:

ADataPtr	Ukazatel na validovanou proměnnou, tj. proměnnou kompatibilní s typem Byte.
AEnumCount	Počet variant výčtového typu.
ALookupFunc	Odkaz na funkci vracející převádějící ordinální hodnotu na ukazatel na znakový řetězec. Viz. poznámky:

Návratové hodnoty:

Konstruktor nevrací žádnou hodnotu.

Poznámky:

4.3.16. Třída TBooleanValidator

Třída **TBooleanValidator** definuje validátor pro výčtový typ Boolean.

```
PBooleanValidator = ^TBooleanValidator;
TBooleanValidator = object( TOrdinalValidator )
public
  FalseText : PString;
  TrueText  : PString;
```

```
    constructor Init( ADataPtr: PBoolean;
                  const AFalseText, ATrueText: string );
    destructor Done; virtual;
    function  TransferText( AText: PString; AMaxLen: Integer;
                AMode: Integer ): Integer; virtual;
    procedure GetMinMax( var AMin, AMax: Longint ); virtual;
    function EnumLookup( AValue: Longint ): PString; virtual;
  end;
```

4.3.16.1. Položka TBooleanValidator.FalseText

Položka **FalseText** obsahuje ukazatel na znakový řetězec obsahující textovou reprezentanci hodnoty False. Položka je inicializována konstruktorem a je určena pouze ke čtení.

```
FalseText : PString;
```

4.3.16.2. Položka TBooleanValidator.TrueText

Položka **TrueText** obsahuje ukazatel na znakový řetězec obsahující textovou reprezentanci hodnoty True. Položka je inicializována konstruktorem a je určena pouze ke čtení.

```
TrueText : PString;
```

4.3.16.3. Konstruktor TBooleanValidator.Init

Konstruktor **Init** provádí inicializaci instance třídy.

```
constructor Init( ADataPtr: PBoolean; const AFalseText,
  ATrueText: string );
```

Parametry:

ADataPtr	Ukazatel na validovanou proměnnou, tj. proměnnou typu Boolean.
AFalseText	Textová reprezentace hodnoty False.
ATrueText	Textová reprezentace hodnoty True.

Návratové hodnoty:

Konstruktor nevrací žádnou hodnotu.

Poznámky:

Konstruktor vytvoří kopii parametrů AFalseText a ATrueText na hromadě a ukazatele na tyto řetězce uloží do položek FalseText a TrueText.

4.3.16.4. Destruktor TBooleanValidator.Done

Destruktor **Done** provádí uvolnění prostředků alokovaných konstruktorem instance.

```
destructor Done; virtual;
```

Parametry:

Destruktor nemá žádné parametry.

Návratové hodnoty:

Destruktor nevrací žádnou hodnotu.

Poznámky:

Destruktor třídy **TBooleanValidator** uvolní z hromady znakové řetězce, na které ukazují položky FalseText a TrueText (viz. kapitoly 4.3.16.1 a 4.3.16.2).

4.3.16.5. Metoda TBooleanValidator.GetMinMax

Metoda **GetMinMax** zjišťuje platný rozsah validované hodnoty.

```
procedure GetMinMax( var AMin, AMax: Longint ); virtual;
```

Parametry:

AMin	Odkaz na proměnnou, do které bude uložena minimální hodnota.
AMax	Odkaz na proměnnou, do které bude uložena maximální hodnota.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Třída **TBooleanValidator** předefinovává metodu **GetMinMax** předka tak, že parametr AMin nastaví na hodnotu 0 a parametru AMax na hodnotu 1.

4.3.16.6. Metoda TBooleanValidator.EnumLookup

Metoda **EnumLookup** vrací ukazatel na název varianty výčtového typu.

```
function  EnumLookup( AValue: Longint ): PString; virtual;
```

Parametry:

AValue	Ordinální hodnota příslušné varianty výčtového typu.

Návratové hodnoty:

Metoda vrací ukazatel na znakový řetězec pro příslušnou variantu výčtového typu. Pokud je parametr AValue mimo rozsah výčtového typu, nebo variantě odpovídá prázdný řetězec, metoda vrací hodnotu **nil**.

Poznámky:

Třída **TBooleanValidator** předefinována metodu **EnumLookup** předka, tak že volá vrací obsah položek FalseText, resp. TrueText.

4.3.16.7. Metoda TBooleanValidator.TransferText

Metoda **TransferText** tvoří jádro validátoru. Provádí převod binární hodnoty svázané s validátorem na textový řetězec a naopak, dále provádí validaci tohoto textového řetězce.

```
function  TransferText( AText: PString; AMaxLen: Integer;
            AMode: Integer ): Integer; virtual;
```

Parametry:

AText	Ukazatel na buffer pro textový řetězec. Délka bufferu je dána parametrem AMaxLen. Tento buffer je určen buď pro zápis nebo pro čtení, podle nastavení parametru AMode.
AMaxLen	Délka bufferu pro textový řetězec, parametr AMaxLen je brán v potaz pouze tehdy, jestliže parametr AMode je nastaven na hodnotu vmLoad. V opačném případě může být hodnota tohoto parametru libovolná.
AMode	Určuje chování metody a může obsahovat tři různé hodnoty:

vmLoad	Metoda provede konverzi binární hodnoty, na kterou ukazuje funkce **GetDataPtr** na textový řetězec.
vmStore	Metoda provede validaci a konverzi textového řetězce na binární hodnotu, na kterou ukazuje funkce **GetDataPtr**.
vmValidate	Metoda provede pouze validaci textového řetězce.

Návratové hodnoty:

Metoda **TransferText** vrací jednu z konstant s prefixem ve_ (viz. kapitola 4.1.2). V případě úspěšného provedení vrací konstantu veOk.

Poznámky:

Třída **TBooleanValidator** předefinována metodu **TransferText** tak, že pro každý parametr AMode různý od vmLoad vrací chybový kód veNotImpl. Metoda provádí pouze převod binární hodnoty na znakový řetězec, pomocí metodu **EnumLookup**.