IoDrv

OBECNÉ OVLADAČE VSTUPNÍCH A VÝSTUPNÍCH ZAŘÍZENÍ TERMINÁLŮ

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: 22.01.2004
(Datum vydání a posledního uložení dokumentu musí být stejné)

Upozornění:
V dokumentu použité názvy výrobků, firem apod. mohou být ochrannými známkami nebo registrovanými ochrannými známkami příslušných vlastníků.

Obsah :

1. O dokumentu

1.1. Revize dokumentu

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

1.2. Účel dokumentu

Tento dokument slouží jako popis knihovny IoDrv**,** 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 je potřeba seznámit se s manuálem Fonts a Bitmaps.

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

Tento manuál je určen především pro programátory ovladačů vstupních a výstupních zařízení nových terminálů. Programátor aplikací by měl prostudovat pouze některé kapitoly, zejména kapitoly týkající se vstupních událostí (4.1, 7.1.1, 7.1.2, 7.1.4, 7.2.1).

3.1. Účel knihovny IoDrv

Knihovna IoDrv definuje obecná rozhraní a společný kód vstupních a výstupních zařízení terminálů firmy SofCon. Jedná se o část balíku vizualizačních knihoven. Obsahuje ovladače klávesnic, myši, dotykových panelů, časovače a displejů pro různá rozlišení a barevné hloubky. Ovladače konkrétních zařízení terminálů jsou v oddělených knihovnách např. T10Drv (Term 10) , T11CDrv (Touch 11 Color) , T33MDrv (Touch 33 Mono) apod.

4. Ovladače vstupních zařízení

4.1. Vstupní události

Každé vstupní zařízení (klávesnice, myš apod.) generuje tzv. vstupní události, které jsou obsluhovány systémem komponent. Vstupní událost je popsána obecnou strukturou **TEvent** definovanou níže:

```
TEvent = record
  Code : Word;                 { Konstanta evXXX              }
  case Integer of
    0: ( WParam   : Word;      { Obecný parametr Word         }
         LParam   : Longint    { Obecný parametr Longint      }
       );
    1: ( Buttons  : Word;      { Stisknutá tlačítka           }
         Pos      : TPoint     { Pozice ukazatele             }
       );
    2: ( KeyCode  : Word;      { Kód klávesy                  }
         VirtKey  : Word;      { Kód virtuální klávesy        }
         CharCode : Char       { ASCII znak                   }
       );
    3: ( TimerId  : Word;      { Identifikátor časovače       }
         Control  : Pointer    { Odkaz na komponentu          }
       );
    4: ( Command  : Word;      { Příkaz obežníku              }
         Param    : Pointer ); { Parametr uživatelské zprávy  }
end;
```

Struktura **TEvent** obsahuje položku **Code**, která jednoznačně určuje typ události. Tato položka může obsahovat jednu z konstant s prefixem ev_. Jejich úplný seznam je uveden v následující tabulce:

Identifikátor	Kód	Popis
evNothing	$0000	Prázdná (neplatná) událost
evKeyDown	$0001	Stisk klávesy
evMouseMove	$0002	Pohyb ukazatele myši
evMouseDown	$0004	Stisk tlačitka myši
evMouseUp	$0008	Uvolnění tlačítka myši
evMouseDbl	$0010	Dvojklik myši
evMouseRep	$0020	Držení tlačítka myši (automatické opakování)
evTimer	$0040	Vypršení limitu časovače
evBroadcast	$0080	Oběžník
evMessage	$0100	Zpráva

Ostatní položky struktury **TEvent** upřesňují další vlastnosti události. Jejich přesný význam je uveden v následujících kapitolách.

Rozdíl mezi událostí typu oběžník je ve zpracování systémem komponent. Událost tpu oběžník je rozeslána všem komponentám ve stromu komponent. Událost typu zpráva je předána vždy pouze jedné komponentě.

4.2. Obecný ovladač vstupů

Obecný ovladač vstupů – třída TInputDriver zapouzdřuje všechny speciální ovladače vstupů (klávesnice, myši, časovače, uživatelské události) a definuje prioritu jednotlivých typů událostí. Třída TInputDriver je definována následovně:

```
TInputDriver = object( TObject )
public
  Keyboard    : PKeyboardDriver;   { Zarizeni klavesnice          }
  Mouse       : PMouseDriver;      { Polohovaci zarizeni          }
  Timer       : PTimerDriver;      { Casovac                      }
  UserQueue   : TEventQueue;       { Fronta uzivatelskych udalosti}

  constructor Init( AKeyboard: PKeyboardDriver; AMouse:
                    PMouseDriver );
  destructor  Done; virtual;

  function  Initialize: Boolean;
  procedure Finalize;
  procedure Tick; virtual;
  function  Broadcast( ACommand: Word; AParam: Pointer ): Boolean;
  function  PutEvent( AEvent: TEvent ): Boolean;
  procedure GetEvent( var AEvent: TEvent; ANoTimer: Boolean );
end;
```

Metoda **GetEvent** vrací nejstarší vstupní událost (dle priority). Po návratu funkce je parametr **AEvent** obsahuje událost. V případě, že žádná událost není k dispozici obsahuje parametr **AEvent** v položce **Code** konstantu evNothing. Priority události definuje následující tabulka:

Priorita	Typ události
Nejvyšší	Uživatelské události
	Události klávesnice

	Události myši (dotykového panelu)
Nejnižší	Událost vypršení limitu časovače

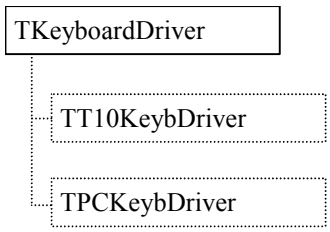
Pomocí metody **PutEvent** lze vložit libovolnou do událost do uživatelské fronty. Podobně metoda **Broadcast** vloží do uživatelské fronty událost typu oběžník (evBroadcast).

Metodu **Tick** je nutné volat v pravidelných intervalech. Obvykle je metoda **Tick** periodicky volána z hlavní smyčky vizulizačního systému. Tato metoda volá stejnojmenné metody **Tick** vložených ovladačů (klávesnice, myš, časovač). Zpravidla z jiné metody **Tick** nadřazeného automatu.

Ke třídě **TInputDriver** nepřistupuje uživatel obvykle přímo, ale skrze metod komponent vizualizačních knihoven. Detailní popis všech třídy **TInputDriver** je uveden v referenční části tohoto manuální v kapitole 7.3.3.

4.3. Ovladač klávesnice

Pokud konkrétní terminál nabízí klávesnici, je v knihovně ovladaču tohoto terminálu vytvořen potomek třídy **TKeyboardDriver**. Tato třída definuje jednotné rozhraní všech typů klávesnic. Všechny metody této třídy jsou deklarovány jako virtuální a potomek je může v případě potřeby předefinovat.



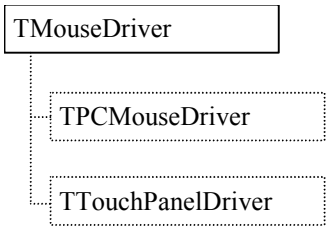

```
TKeyboardDriver = object( TObject )
public
  function  Initialize: Boolean; virtual;
  procedure Finalize; virtual;

  procedure Tick; virtual;
  procedure GetEvent( var Event: TEvent ); virtual;

  procedure SetRepeatDelay( AValue: Integer ); virtual;
  function  GetRepeatDelay: Integer; virtual;

  procedure SetRepeatRate( AValue: Integer ); virtual;
  function  GetRepeatRate: Integer; virtual;

  function  SetSettings( const ASettings: TKeyboardSettings ):
              Boolean; virtual;
  procedure GetSettings( var ASettings: TKeyboardSettings );
              virtual;
end;
```

Metoda **GetEvent** vrací nejstarší vstupní událost, tj. kód první stisknuté klávesu, která ještě nebyla zpracována. Parametr **AEvent** je naplněn tak, že položka **Code** obsahuje konstantu evKeyDown. Pokud byla stisknuta některá ze speciálních kláves (např. šipka vlevo), pak je položka **KeyCode** je naplněna příslušnou konstantou kbXXX (viz. kapitola 7.1.2). Pokud byla stisknuta klávesa reprezentující znak z ASCII, pak je příslušný kód znaku uložen do položky **CharCode** a nižšího bajtu **KeyCode**, vyšších osm bitů **KeyCode** je nastaveno na nulu.

Ostatní metody s prefixem **Set_** a **Get_** slouží k nastavení některých typických vlastností klávesnice.

Detailní popis třídy **TKeyboardDriver** je uveden v referenční části manuálu v kapitole 7.3.4.

4.4. Ovladač myši a dotykového panelu



Pokud konkrétní terminál nabízí nějaké polohovací zařízení (myš, dotykový panel), je v knihovně ovladaču tohoto terminálu vytvořen potomek třídy **TMouseDriver**. Tato třída definuje jednotné rozhraní všech typů polohovacích zařízení. Téměř všechny metody této třídy jsou deklarovány jako virtuální a potomek je může v případě potřeby předefinovat.

```
TMouseDriver = object( TObject )
public
  DispDriver : PDisplayDriver;

  function  Initialize: Boolean; virtual;
  procedure Finalize; virtual;

  procedure Tick; virtual;
  procedure GetEvent( var Event: TEvent ); virtual;

  procedure SetDisplayDriver( ADisplayDriver: PDisplayDriver );

  procedure SetMode( ACalibration: Boolean ); virtual;

  function  Calibrate( DispA, MouseA, DispB, MouseB,
              DispC, MouseC : TPoint ): Boolean; virtual;

  procedure SetPressDelay( AValue: Byte ); virtual;
  function  GetPressDelay: Byte; virtual;
  procedure SetReleaseDelay( AValue: Byte ); virtual;
  function  GetReleaseDelay: Byte; virtual;
  procedure SetSensitivity( AValue: Byte ); virtual;
  function  GetSensitivity: Byte; virtual;
  procedure SetFilterWeight( AValue: Byte ); virtual;
  function  GetFilterWeight: Byte; virtual;
  procedure SetStabilization( AValue: Byte ); virtual;
  function  GetStabilization: Byte; virtual;
  procedure SetDblClickDelay( AValue: Integer ); virtual;
  function  GetDblClickDelay: Integer; virtual;
  procedure SetDblClickArea( AValue: Integer ); virtual;
  function  GetDblClickArea: Integer; virtual;
  procedure SetRepeatDelay( AValue: Integer ); virtual;
  function  GetRepeatDelay: Integer; virtual;
  procedure SetRepeatRate( AValue: Integer ); virtual;
  function  GetRepeatRate: Integer; virtual;
  procedure GetCalibration( var ACalibration:
              TMouseCalibration ); virtual;
  procedure SetCalibration( const ACalibration:
              TMouseCalibration ); virtual;

  procedure GetSettings( var ASettings: TMouseSettings ); virtual;
```

```
    function  SetSettings( const ASettings:
                TMouseSettings ): Boolean; virtual;
  end;
```

Metoda **GetEvent** vrací nejstarší vstupní událost. Ovladač polohovacího zařízení může generovat následující typy událostí. Typ události je rozlišen položkou **Code** parametru **AEvent**:

Identifikátor	Událost
evMouseMove	Změna polohy ukazatele.
evMouseDown	Stisk tlačítka myši (stisk dotykového panelu)
evMouseUp	Uvolnění tlačítka myši (uvolnění dotykového panelu)
evMouseDbl	Dvojklik tlačítka myši (dotykového panelu)
evMouseRep	Držení tlačítka myši (dotykového panelu), událost je opakována s nastavenou periodou po nastavené době od stisku.

Při všechny výše uvedených událostech jsou vyplněny navíc položky **Buttons** a **Pos** parametru **AEvent**. Položka **Buttons** obsahuje kombinaci konstant mbXXX, tj. kombinaci stisknutých tlačítek myši (při stisku dotykového panelu je emulován stisk levého tlačítka myši, tj. konstanta mbLeft). Položka **Pos** je naplněna souřadnicemi ukazatele, ve kterých událost nastala.

Ostatní metody s prefixem **Set_** a **Get_** slouží k nastavení některých typických vlastností myši příp. dotykového panelu. Potomci třídy **TMouseDriver** nemusí využívat všechny tyto funkce.

Detailní popis třídy **TMouseDriver** je uveden v referenční části manuálu v kapitole 7.3.4.

4.5. Ovladač časovače

Ovladač časovače umožňuje naplánování jednorázové příp. periodické časové události. Ovladač časovače je definován třídou **TTimerDriver**:

```
  TTimerDriver = object( TObject )
  public
    constructor Init( AMaxTimers: Integer );
    destructor  Done; virtual;

    procedure GetEvent( var Event: TEvent ); virtual;

    function  Schedule( AControl: Pointer; AID: Integer;
                ATime: Longint; AOptions: Word ): Boolean;
    procedure Cancel( AControl: Pointer; AId: Integer );
  end;
```

Pomocí metody **Schedule** lze naplánovat událost s přesností danou intervalem přerušení časovače IRQ 0 (tj. obvykle 55ms). Celkem lze naplánovat současně až 16 událostí. Každá událost má přidělen identifikátor (parametr Id) a může být svázána s konkrétní instancí libovolné komponenty (parametr AControl).

Naplánované události lze předčasně zrušit pomocí metody **Cancel**.

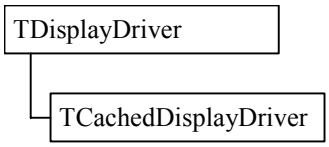
Metoda **GetEvent** vrací první událost, u které již vypršel časový limit. Položka Code parametru **AEvent** je nastavena na hodnotu evTimer. Položky **TimerId** a **Control** jsou obsahují parametry naplánované události. Pokud před voláním metody GetEvent nevypršel limit žádného naplánovaného časovače, pak položka **Code** parametru **AEvent** obsahuje konstantu **evNothing**.

Detailní popis třídy **TTimerDriver** je uveden v referenční části manuálu v kapitole 7.3.6.

5. Ovladače výstupních zařízení

5.1. Ovladač displeje

Ovladače všech typů displejů vycházejí z abstraktní třídy **TDisplayDriver**. Ilustrativní deklarace této třídy je uvedena níže:




```
TDisplayDriver = object( TObject )
public
  Width       : Integer;        { Pocet bodu na radek            }
  Height      : Integer;        { Pocet sloupcu                  }
  BPP         : Byte;           { Pocet bitu na pixel            }
  Surface     : PDrawSurface;   { Kreslici povrch                }
  Caret       : PCaret;         { Stav textoveho kurzoru         }
  Cursor      : PCursor;        { Stav kurzoru mysi              }

  . . .

  constructor Init( AWidth, AHeight: Integer; ABPP: Byte );
  destructor  Done; virtual;

  function  Initialize: Boolean; virtual;
  procedure Finalize; virtual;

  procedure BeginDraw;
  procedure EndDraw;

  procedure SurfaceChanged; virtual;

  function  GetContrast: Integer; virtual;
  procedure SetContrast( AValue: Integer ); virtual;
  function  GetBrightness: Integer; virtual;
  procedure SetBrightness( AValue: Integer ); virtual;
  function  GetAutoOffDelay: Integer; virtual;
  procedure SetAutoOffDelay( AValue: Integer ); virtual;
  function  GetAutoOffBrightness: Integer; virtual;
  procedure SetAutoOffBrightness( AValue: Integer ); virtual;

  procedure GetSettings( var ASettings:
              TDisplaySettings ); virtual;
  function  SetSettings( const ASettings:
              TDisplaySettings ): Boolean; virtual;
```

```
    function  Wakeup: Boolean;

    procedure Tick; virtual;
  end;
```

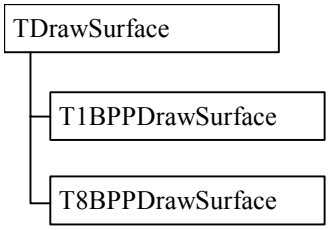
Ovladač displeje zapouzdřuje další pomocné ovladače pro textový kurzor a ukazatel myši. Položka instance Surface se odkazuje na tzv. kreslící povrch (tj. instance třídy **TDrawSurface** poskytující funkce pro zápis (kreslení) do video paměti různých typů displejů)

Detailní popis třídy **TDisplayDriver** je uveden v referenční části manuálu v kapitole 7.3.7.

Z třídy **TDisplayDriver** je přímo odvozena třída **TCachedDisplayDriver**, která je určena pro displeje s video-pamětí, která není přímo mapovaná v paměťovém prostoru procesoru. Třída **TCachedDisplayDriver** obsahuje vyrovnávací paměť (kopii videopaměti), která je ve vhodných okamžicích přepisována do paměti řadiče displeje (např. po sběrnici IOBUS nebo po sériové lince). Detailní popis třídy **TCachedDisplayDriver** je uveden v referenční části manuálu v kapitole 7.3.8.

5.2. Kreslící povrch

Kreslící povrch je zobecněním videopaměti displejů. Obecné rozhraní kreslících povrchů definuje abstraktní třída **TDrawSurface.** Tato třída nabízí množství metod pro zápis do videopaměti (vykreslování grafických objektů).




```
  TDrawSurface = object( TObject )
  public
    Changes    : TRect;       { Obdelnik ohranicujici zmeny         }

    constructor Init;
    destructor Done; virtual;

    function  ClipLine( const AParams: TDrawParams;
                var AResult: TLineBres ): Boolean;

    function  Initialize: Boolean; virtual;
    procedure Finalize; virtual;

    function  Clear( const AParams: TDrawParams ): TDrawStatus;
                virtual;
    function  Copy( const AParams: TDrawParams ): TDrawStatus;
                virtual;
    function  Scroll( const AParams: TDrawParams ): TDrawStatus;
                virtual;
    function  DrawText( const AParams: TDrawParams ): TDrawStatus;
                virtual;

    function  DrawPoint( const AParams: TDrawParams ): TDrawStatus;
                virtual;
    function  FillRect( const AParams: TDrawParams ): TDrawStatus;
                virtual;
```

```
    function  DrawBitmap( const AParams: TDrawParams ): TDrawStatus;
                virtual;

    function  DrawLine( const AParams: TDrawParams ): TDrawStatus;
                virtual;
    function  DrawCircle( const AParams: TDrawParams ): TDrawStatus;
                virtual;
    function  DrawEllipse( const AParams: TDrawParams ): TDrawStatus;
                virtual;
    function  DrawArc( const AParams: TDrawParams ): TDrawStatus;
                virtual;
    function  FillCircle( const AParams: TDrawParams ): TDrawStatus;
                virtual;
    function  FillEllipse( const AParams: TDrawParams ): TDrawStatus;
                virtual;
    function  FillArc( const AParams: TDrawParams ): TDrawStatus;
                virtual;
    function  CopyToBitmap( const AParams: TDrawParams ):
                TDrawStatus; virtual;

  end;
```

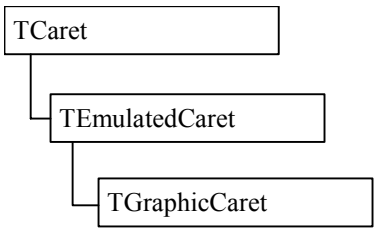
Detailní popis třídy **TDrawSurface** je uveden v referenční části manuálu v kapitole 7.3.9.

Z abstraktní třídy TDrawSurface jsou odvozeny další třídy pro konkrétní typy kreslících povrchů. Přímo v knihovně IoDrv jsou definování tito potomci:

T1BPPDrawSurface pro kreslící povrchy s 1 bitem na pixel (viz. kapitola 7.3.10)
T8BPPDrawSurface pro kreslící povrchy s 8 bity na pixel. (viz. kapitola 7.3.10)

5.3. Textový kurzor

Textový kurzor je malý blikající obdélník v místě vkládání nebo přepisování znaku zpravidla v editační řádce. Obecné rozhraní textového kurzoru je definováno abstraktní třídou **TCaret**.




```
  TCaret = object( TObject )
  public
    procedure Show( APosition, ASize: TPoint;
                    const AClip: TRect ); virtual;
    procedure Hide; virtual;
    procedure SetBlinkPeriod( APeriod: Integer ); virtual;

    ...
    procedure Tick; virtual;
  end;
```

Detailní popis třídy **TCaret** je uveden v referenční části manuálu v kapitole 7.3.14.

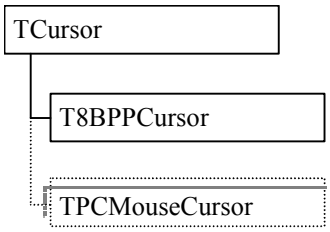
Z abstraktní třídy **TCaret** je odvozena abstraktní třída **TEmulatedCaret**. Jedná se o emulovaný textový kurzor. Potomci **TEmulatedCaret** musí předefinovat metodu **DrawCaret**, tak aby správně vykreslovala obdélník textového kurzoru.

```
TEmulatedCaret = object( TCaret )
public
  procedure DrawCaret( AShow: Boolean ); virtual;
end;
```

Detailní popis třídy **TEmulatedCaret** je uveden v referenční části manuálu v kapitole 7.3.15.

Pro terminály s grafickým displejem je dále definována třída **TGraphicCaret**, která je potomkem **TEmulatedCaret**. Tato třída zcela implementuje textový kurzor pro tyto typy displejů. Detailní popis třídy **TGraphicCaret** je uveden v referenční části manuálu v kapitole 7.3.16.

5.4. Ukazatel myši



Ukazatel myši je malá šipka zobrazovaná v aktuálních souřadnicích myši. Obecné rozhraní ukazatele myši je definováno pomocí abstraktní třídy **TCursor**. Instance třídy ukazatele myši se vytváří jen v případě, že příslušný terminál je vybaven polohovacím zařízení jako je myš, trackball apod. (např. simulátor terminálu na PC). Běžné terminály fy SofCon ukazatel myši nezobrazují.

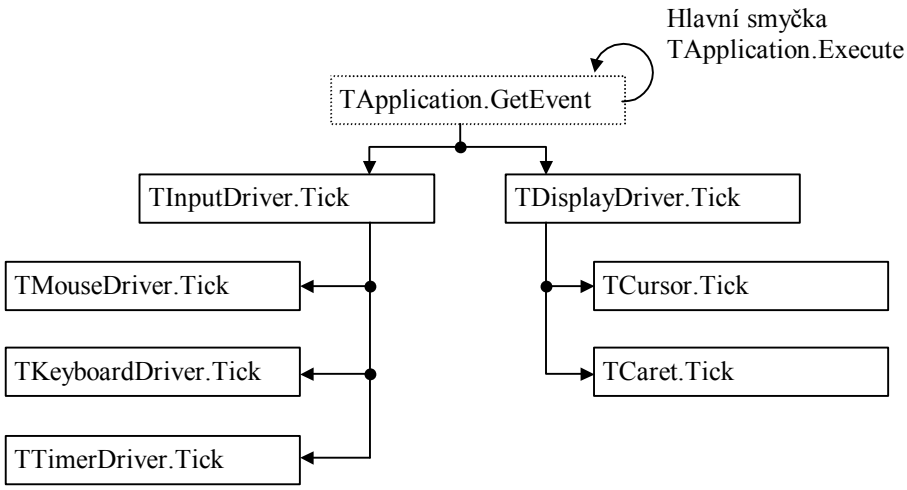
```
TCursor = object( TObject )
public
  procedure SetPosition( AX, AY: Integer ); virtual;
  procedure Show; virtual;
  procedure Hide; virtual;
end;
```

Detailní popis třídy **TCursor** je uveden v referenční části manuálu v kapitole 7.3.12.

Pomocí třídy **TCursor** jsou definovány konkrétní ukazatele myši, např. **T8BPPCursor** (popsaný v kapitole 7.3.13) implementuje ukazatel myši pro kreslící povrchy s 8 bity na pixel, nebo **TPCMouseCursor** (definovaný v knihovně PCDrv) implementuje rozhraní pro standardní ukazatel myši realizovaný pomocí rezidentního ovladače myši (služba Int33).

6. Stavový automat ovladačů a metoda Tick.

Všechny ovladače vstupů a výstupů jsou obecně implementovány jako stavový automat. Abstraktní rozhraní ovladačů vždy obsahuje metodu nazvanou **Tick**. Tato metoda provádí jeden krok automatu a musí být volána v pravidelných intervalech (obvykle cca 50ms). Na následujícím obrázku je znázorněn strom volání jednotlivých metod **Tick**. Pořadí volání je v podstatě určeno propojením instancí tříd ovladačů. Metoda **Tick** ovladače displeje a ovladače vstupů jsou vyvolány v hlavní smyčce vizualizační stavebnice pomocí třídy **TApplication** (viz. knihovna Controls).



7. Reference

7.1. Konstanty

7.1.1. Konstanty evXXX

Konstanty s prefixem ev_ slouží jako identifikátory událostí vstupních zařízení. Vždy jedna z těchto konstant je uložena v položce Code struktury **TEvent** (viz. kapitola 7.2.1).

Identifikátor	Kód	Popis
evNothing	$0000	Prázdná (neplatná) událost
evKeyDown	$0001	Stisk klávesy
evMouseMove	$0002	Pohyb ukazatele myši
evMouseDown	$0004	Stisk tlačítka myši (stisk dotykového panelu)
evMouseUp	$0008	Uvolnění tlačítka myši
evMouseDbl	$0010	Dvojklik myši
evMouseRep	$0020	Držení tlačítka myši (automatické opakování)
evTimer	$0040	Vypršení limitu časovače
evBroadcast	$0080	Oběžník
evMessage	$0100	Zpráva

7.1.2. Konstanty kbXXX

Klávesy (příp. kombinace kláves) terminálů jsou identifikovány jednoznačným kódem – konstantou s prefixem kb_. Seznam těchto konstant je uveden v tabulce níže. Ostatní klávesy, které nejsou uvedeny v této tabulce (např. klávesy A až Z) mají kódy shodné s kódy ASCII sady.

Terminál (resp. ovladač klávesnice terminálu, viz. kapitola 4.3) generuje pouze ty

kódy kláves jimiž je terminál vybaven. Kód klávesy je ovladačem klávesnice uložen do položky KeyCode parametru AEvent při volání metody **GetEvent**.

Identifikátor	Kód	Klávesa
kbEsc	$001B	Esc
kbEnter	$000D	Enter
kbBackSpace	$0008	Backspace
kbTab	$0009	Tab
kbLeft	$0103	Šipka vlevo
kbRight	$0104	Šipka vpravo
kbUp	$0105	Šipka nahoru
kbDown	$0106	Šipka dolu
kbInsert	$0107	Insert
kbDelete	$0108	Delete
kbHome	$010A	Home
kbEnd	$010B	End
kbPageUp	$010C	Page Up
kbPageDown	$010D	Page Down
kbStart	$010E	Start
kbStop	$010F	Stop
kbShiftTab	$0110	Shift + Tab
kbShiftEnter	$0111	Shift + Enter
kbShiftLeft	$0112	Shift + Šipka vlevo
kbShiftRight	$0113	Shift + Šipka vpravo
kbAlt	$0116	Alt
kbClear	$0117	Clear
kbF1	$0120	F1
kbF2	$0121	F2
kbF3	$0122	F3
kbF4	$0123	F4
kbF5	$0124	F5
kbF6	$0125	F6
kbF7	$0126	F7
kbF8	$0127	F8
kbF9	$0128	F9
kbF10	$0129	F10
kbF11	$012A	F11
kbF12	$012B	F12
kbShiftF1	$0130	Shift + F1
kbShiftF2	$0131	Shift + F2
kbShiftF3	$0132	Shift + F3
kbShiftF4	$0133	Shift + F4
kbShiftF5	$0134	Shift + F5
kbShiftF6	$0135	Shift + F6
kbShiftF7	$0136	Shift + F7
kbShiftF8	$0137	Shift + F8
kbShiftF9	$0138	Shift + F9

kbShiftF10	$0139	Shift + F10
kbShiftF11	$013A	Shift + F11
kbShiftF12	$013B	Shift + F12
kbCtrlF1	$0140	Ctrl + F1
kbCtrlF2	$0141	Ctrl + F2
kbCtrlF3	$0142	Ctrl + F3
kbCtrlF4	$0143	Ctrl + F4
kbCtrlF5	$0144	Ctrl + F5
kbCtrlF6	$0145	Ctrl + F6
kbCtrlF7	$0146	Ctrl + F7
kbCtrlF8	$0147	Ctrl + F8
kbCtrlF9	$0148	Ctrl + F9
kbCtrlF10	$0149	Ctrl + F10
kbCtrlF11	$014A	Ctrl + F11
kbCtrlF12	$014B	Ctrl + F12
kbAltF1	$0150	Alt + F1
kbAltF2	$0151	Alt + F2
kbAltF3	$0152	Alt + F3
kbAltF4	$0153	Alt + F4
kbAltF5	$0154	Alt + F5
kbAltF6	$0155	Alt + F6
kbAltF7	$0156	Alt + F7
kbAltF8	$0157	Alt + F8
kbAltF9	$0158	Alt + F9
kbAltF10	$0159	Alt + F10
kbAltF11	$015A	Alt + F11
kbAltF12	$015B	Alt + F12
kbAltA	$0160	Alt + A
kbAltB	$0161	Alt + B
kbAltC	$0162	Alt + C
kbAltD	$0163	Alt + D
kbAltE	$0164	Alt + E
kbAltF	$0165	Alt + F
kbAltG	$0166	Alt + G
kbAltH	$0167	Alt + H
kbAltI	$0168	Alt + I
kbAltJ	$0169	Alt + J
kbAltK	$016A	Alt + K
kbAltL	$016B	Alt + L
kbAltM	$016C	Alt + M
kbAltN	$016D	Alt + N
kbAltO	$016E	Alt + O
kbAltP	$016F	Alt + P
kbAltQ	$0170	Alt + Q
kbAltR	$0171	Alt + R
kbAltS	$0172	Alt + S
kbAltT	$0173	Alt + T

kbAltU	$0174	Alt + U
kbAltV	$0175	Alt + V
kbAltW	$0176	Alt + W
kbAltX	$0177	Alt + X
kbAltY	$0178	Alt + Y
kbAltZ	$0179	Alt + Z
kbAlt0	$017A	Alt + 0
kbAlt1	$017B	Alt + 1
kbAlt2	$017C	Alt + 2
kbAlt3	$017D	Alt + 3
kbAlt4	$017E	Alt + 4
kbAlt5	$017F	Alt + 5
kbAlt6	$0180	Alt + 6
kbAlt7	$0181	Alt + 7
kbAlt8	$0182	Alt + 8
kbAlt9	$0183	Alt + 9

7.1.3. Konstanty vkXXX

Konstanty s prefixem vk_ slouží jako identifikátory virtuálních kláves. Více v dokumentaci ke knihovně Controls.

Identifikátor	Hodnota	Popis
vkBase	$200	Definuje bázovou hodnotu konstant vk_

V této knihovně je definována konstanta vkBase, která určuje bázovou hodnotu pro všechny další konstanty vk_ (, které jsou definovány v knihovně Controls).

7.1.4. Konstanty mbXXX

Konstanty s prefixem mb_ slouží k identifikaci tlačítek myši. Kód tlačítka myši ukládá ovladač myši **TMouseDriver** (viz. kapitola 7.3.5) do položky Buttons parametr AEvent při volání metody **GetEvent**. Ovladače dotykových panelů emulují při stisku stisk levého tlačítka myši.

Identifikátor	Hodnota	Tlačítko
mbLeft	$01	Levé
mbRight	$02	Pravé
mbCenter	$04	Prostřední

7.1.5. Konstanty clXXX

Ovladače displejů umožňují pracovat zároveň až s 256 barvami. Každá barva má jednoznačný identifikátor s prefixem cl_. V knihovně IoDrv jsou definovány pouze dvě základní barvy a to černá a bílá.

Identifikátor	Hodnota	Barva
clBlack	0	Černá
clWhite	15	Bílá

7.1.6. Konstanty fsXXX

Konstanty s prefixem fs_ upřesňují chování metody **DrawText** (viz. kapitola 7.3.9.10) třídy **TDrawSurface**. Konstanta se ukládá do položky Style struktury **TFontRef**, která definuje použité písmo. (viz. kapitola 7.2.3)

Identifikátor	Hodnota	Popis
fsTransparent	$01	Transparentní písmo
*fsBold**	*$02*	*Tučné písmo*
*fsItalic**	*$04*	*Kurzíva*
*fsDisabled**	*$08*	*Písmo s polovičním jasem*

* neimplementováno

7.1.7. Konstanty bsXXX

Konstanty s prefixem bs_ upřesňují styl vyplňovaných ploch. Konstanta se ukládá do položky Style struktury **TBrushRef**, která definuje použité písmo. (viz. kapitola 7.2.4)

Identifikátor	Hodnota	Popis
bsSolid	0	Jednobarevný štětec
*bsPattern**	*1*	*Vzorovaný štětec*

* neimplementováno

7.1.8. Konstanty psXXX

Konstanty s prefixem ps_ upřesňují styl vykreslovaných čar. Konstanta se ukládá do položky Style struktury **TPenRef**, která definuje použité písmo. (viz. kapitola 7.2.5)

Identifikátor	Hodnota	Popis
psSolid	0	Plná čára
psDotted	1	Tečkovaná čára
psDashed	2	Přerušovaná čára

7.1.9. Konstanty ropXXX

Konstanty s prefixem rop_ upřesňují chování vykreslovacích funkcí třídy **TDrawSurface**. Definují logickou operaci barvy zapisovaných bodů s jejich původní barvou (viz. položka ROP struktury **TDrawParams** v kapitole 7.2.7)

Identifikátor	Hodnota	Popis
ropCopy	0	Nahrazení barvy pozadí zapisovaným barvy
ropOr	1	Logický součet zapisované barvy s barvou pozadí
ropAnd	2	Logický součin zapisovaného barvy s barvou pozadí
ropAndNot	3	Logický součin negace zapisovaného barvy s barvou pozadí
ropXor	4	Nonekvivalence zapisovaného barvy a barvy pozadí

7.1.10. Konstanty dsXXX

Konstanty s prefixem ds_ slouží jako návratové kódy vykreslovacích funkcí třídy **TDrawSurface**, jejichž návratový typ je **TDrawStatus**.

Identifikátor	Hodnota	Popis
dsSuccess	0	Funkce proběhla v pořádku
dsNotImpl	1	Funkce není implementovaná
dsInvPar	2	Neplatné parametry funkce

7.1.11. Konstanty toXXX

Konstanty s prefixem to_ upřesňují vlastnosti časovače naplánovaného pomocí třídy **TTimerDriver** (viz. parametr AOptions metody **Schedule** třídy **TTimerDriver**).

Identifikátor	Hodnota	Popis
toOneShort	0	Jednorázový časovač
toPeriodic	1	Periodický časovač
*toAtTime**	*2*	

* neimplementováno

7.2. Typy

7.2.1. TEvent

Univerzální struktura **TEvent** popisuje událost vstupního zařízení.

```
TEvent = record
  Code : Word;                  { Konstanta evXXX              }
  case Integer of
    0: ( WParam   : Word;       { Obecný parametr Word         }
         LParam   : Longint     { Obecný parametr Longint      }
       );
    1: ( Buttons  : Word;       { Stisknutá tlačítka           }
         Pos      : TPoint      { Pozice ukazatele             }
       );
    2: ( KeyCode  : Word;       { Kód klávesy                  }
```

```
          VirtKey  : Word;      { Kód virtuální klávesy        }
          CharCode : Char       { ASCII znak                   }
        );
      3: ( TimerId  : Word;      { Identifikátor časovače       }
          Control  : Pointer    { Odkaz na komponentu          }
        );
      4: ( Command  : Word;      { Příkaz oběžníku              }
          Param    : Pointer ); { Parametr uživatelské zprávy  }
end;
```

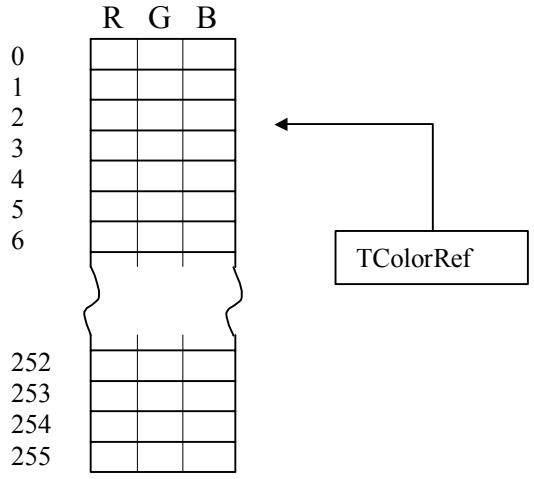
Význam jednotlivých položek je uveden v následující tabulce:

Položka	Popis
Code	Položka Code obsahuje jednu z konstant s prefixem ev_ (viz kapitola 7.1.1) a určuje typ události. Položka Code určuje zároveň platnost ostatních položek struktury.
WParam	Obecný parametr typu WORD
LParam	Obecný parametr typu LONGINT
Buttons	Stisknutá tlačítka myši (evMouseXXX)
Pos	Pozice ukazatele myši (evMouseXXX)
KeyCode	Kód klávesy kb_ (evKeyDown)
VirtKey	Kód virtuální klávesy vk_ (evKeyDown)
CharCode	ASCII znak (evKeyDown)
TimerId	Identifikátor časovače (evTimer)
Control	Odkaz na komponentu (evTimer)
Command	Příkaz (evBroadcast, evMessage)
Param	Parametr příkazu (evBroadcast, evMessage)

7.2.2. TColorRef

Typ **TColorRef** definuje index do palety barev, tedy do R-G-B tabulky, jejíž délka je dána barevným rozlišením displeje. Maximální počet podporovaných barev je 256 (což odpovídá 8 bitům na pixel).

TColorRef = Byte;



Implicitní nastavení RGB tabulky je poplatné konkrétnímu displeji a je uvedeno v příslušném manuálu. Obecně se lze spolehnout pouze na indexy 0 (konstanta clBlack) a 15 (konstanta clWhite), které reprezentují černou a bílou barvu.

7.2.3. TFontRef

Struktura **TFontRef** definuje písmo. Definice písma se používá při vykreslování textu pomocí metody **DrawText** třídy **TDrawSurface**.

```
TFontRef = record
  Style   : Byte;
  Color   : TColorRef;
  BkColor : TColorRef;
  Leading : Byte;
  Id      : Word;
end;
```

Popis jednotlivých položek je uveden v následující tabulce:

Položka	Popis
Style	Položka Style určuje styl písma a může obsahovat kombinaci konstant s prefixem fs_ (viz. kapitola 7.1.6)
Color	Položka Color určuje barvu znaků a může obsahovat hodnotu od 0 do 255 nebo konstantu s prefixem cl_ (viz. kapitola 7.1.5).
BkColor	Položka BkColor určuje barvu pozadí znaků a může obsahovat hodnotu od 0 do 255 nebo konstantu s prefixem cl_ (viz. kapitola 7.1.5). Tato barva se uplatní v případě, že položka Style neobsahuje příznak fsTransparent.
Leading	
Id	Položka Id obsahuje identifikátor registrovaného fontu pomocí funkcí knihovny Fonts. Jedná se o identifikátor s prefixem fid_.

7.2.4. TBrushRef

Struktura **TBrushRef** definuje tzv. štětec, tj. jeho styl, barvu, vzor a případně jeho počátek. Štětec se používá při vykreslování vyplněných objektů, jako je vyplněný obdelník, kruh apod.

```
TBrushRef = record
  Style  : Byte;
  Color  : TColorRef;
  Origin : TPoint;
  Bitmap : TBitmap;
end;
```

Popis jednotlivých položek je uveden v následující tabulce:

Položka	Popis
Style	Položka Style určuje styl štětce a může obsahovat jednu

	z konstant s prefixem bs_ (viz. kapitola 7.1.7)
Color	Položka Color určuje barvu štětce a může obsahovat hodnotu od 0 do 255 nebo konstantu s prefixem cl_ (viz. kapitola 7.1.5).
Bitmap	Položka Bitmap specifikuje bitmapu vzoru, kterým bude plocha vyplněna v případě, že položka Style obsahuje konstantu bsPattern. *Zatím neimplementováno.*
Origin	Položka Origin určuje posunutí bitmapy vzoru v rámci vyplňované plochy. *Zatím neimplementováno.*

7.2.5. TPenRef

Struktura **TPenRef** definuje tzv. pero, tj. jeho styl, barvu a šířku. Pero se používá při vykreslování čárových objektů, jako je bod, úsečka, kružnice, elipsa apod.

```
TPenRef = record
  Style : Byte;
  Color : TColorRef;
  Width : Byte;
{ _Res    : Byte; }
end;
```

Popis jednotlivých položek je uveden v následující tabulce:

Položka	Popis
Style	Položka Style určuje styl pera a může obsahovat jednu z konstant s prefixem ps_ (viz. kapitola 7.1.8)
Color	Položka Color určuje barvu pera a může obsahovat hodnotu od 0 do 255 nebo konstantu s prefixem cl_ (viz. kapitola 7.1.5).
Width	Položka Width určuje šířka pera v pixelech. Může nabývat hodnot 1 až 8.

7.2.6. TDrawParamPoints

Type **TDrawParamsPoints** je interní typ použitý ve struktuře **TDrawParams** (viz. kapitola 7.2.6).

```
TDrawParamPoints = array[0..3] of TPoint;
```

7.2.7. TDrawParams

Struktura **TDrawParams** slouží jako jediný univerzální parametr vykreslovacích metod třídy **TDrawSurface**. Význam jednotlivých položek je závislý na konkrétní volané metodě.

```
TDrawParams = record
    Points : TDrawParamPoints;
    Clip   : TRect;
    ROP    : Byte;
    Option : Byte;

    case Integer of
```

```
      0 : ( Font   : TFontRef;
            Text   : PChar;
            Length : Word;
          );

      1 : ( Bitmap : PBitmap );
      2 : ( Pen    : TPenRef;
            Brush  : TBrushRef;
          );
  end;
```

Položka	Popis
Points	Položka Points je pole čtyř bodů v souřadnicích displeje, které určují rozměry vykreslovaného objektu.
Clip	Položka Clip určuje obdélník v souřadnicích displeje, podle kterého bude vykreslovaný objekt oříznut. Pouze ta část vykreslovaného objektu, která se nachází v tomto obdélníku bude vykreslena.
ROP	Položka ROP určuje logickou operaci barvy vykreslovaných bodů s barvou pozadí. Jedná se o jednu z konstant s prefixem rop_ (viz. kapitola 7.1.9).
Option	Položka Option je volitelná a vykreslovací funkce ji mohou interpretovat naprosto libovolně.
Font	Položka Font definuje parametry písma (viz. kapitola 7.2.3), kterým je vykreslen text z položky Text.
Text	Ukazatel na vykreslovaný text. Text může být zakončen nulovým znakem nebo může být omezen délkou danou položkou Length.
Length	Počet znaků vykreslovaného textu.
Bitmap	Položka Bitmap obsahuje ukazatel na vykreslovanou bitmapu.
Pen	Položka Pen definuje pero použité pro vykreslování čárových objektů (viz. kapitola 7.2.5)
Brush	Položka Brush definuje štětec použitý pro vykreslování vyplněných objektů (viz. kapitola 7.2.4)

7.2.8. TDrawStatus

Type **TDrawStatus** je návratový typ vykreslovacích metod třídy **TDrawSurface**. Návratové kódy jsou konstanty s prefixem ds_ (viz. kapitola 7.1.10).

```
TDrawStatus = Integer;
```

7.2.9. TLineBres

Struktura **TLineBres** je interní struktura používaná pro předání parametrů pro rasterizaci úsečky bresenhamovým algoritmem. Struktura **TLineBres** je výstupem metody **ClipLine** třídy **TDrawSurface**.

```
TLineBres = record
  X1     : Integer;
  Y1     : Integer;
  Pixels : Integer;
  DeltaX : Integer;
```

```
    DeltaY : Integer;
    D      : Integer;
    DInc1  : Integer;
    DInc2  : Integer;
    XInc1  : Integer;
    XInc2  : Integer;
    YInc1  : Integer;
    YInc2  : Integer;
  end;
```

7.2.10. TEventArray

Typ **TEventArray** je spíše interní struktura pro implementaci fronty událostí ve třídě **TEventQueue**.

```
  PEventArray = ^TEventArray;
  TEventArray = array[0..$FFF8 div SizeOf( TEvent ) - 1] of TEvent;
```

7.2.11. TKeyboardSettings

Struktura **TKeyboardSettings** popisuje obecné nastavení klávesnice. Tuto strukturu využívají metody **GetSettings** a **SetSettings** třídy **TKeyboardDriver** (viz. kapitoly 7.3.4.10, 7.3.4.9)

```
  TKeyboardSettings = record
    Size          : Word;
    CheckSum      : Word;
    RepeatDelay   : Integer;
    RepeatRate    : Integer;
  end;
```

Jednotlivé položky struktury popisuje následující tabulka:

Položka	Popis
Size	Položka Size obsahuje délku struktury, tj. SizeOf(TKeyboardSettings).
CheckSum	Položka CheckSum obsahuje kontrolní součet celé struktury. Kontrolní součet je počítán pomocí funkce **GetCheckSum** (viz. kapitola 7.4.3).
RepeatDelay	Položka RepeatDelay určuje zpoždění před automatickým opakováním stisku klávesy. Hodnota je zadávána v milisekundách.
RepeatRate	Položka RepeatRate určuje periodu automatického opakování stisku klávesy. Hodnota je zadávaná v milisekundách.

7.2.12. TMouseCalibration

Struktura **TMouseCalibration** obsahuje konstanty pro převod souřadnic polohovacího zařízení (zejména dotykového panelu) na souřadnice displeje.

```
  TMouseCalibration = record
    XMultiplier   : Integer;     { Nasobici konstanta pro osu X  }
```

```
    XAddend       : Integer;     { Pricitaci konstanta pro osu X }
    YMultiplier   : Integer;     { Nasobici konstanta pro osu Y  }
    YAddend       : Integer;     { Pricitaci konstanta pro osu Y }
    SwapXY        : Boolean;     { Prohozeni souradnic X a Y     }
  end;
```

Význam jednotlivých položek je zřejmý z následujících převodních vztahů:

$$Xd = \left\lfloor \frac{Xi \cdot XMultiplier}{4096} \right\rfloor + XAddend$$

$$Yd = \left\lfloor \frac{Yi \cdot YMultiplier}{4096} \right\rfloor + YAddend$$

kde [Xi, Yi] je bod v souřadnicích polohovacího zařízení
[Xd, Yd] je bod v souřadnicích displeje

V případě, že je položka **SwapXY** nastavena na hodnotu True, pak jsou souřadnice Xd a Yd provedení výpočtu prohozeny.

7.2.13. TMouseSettings

Struktura **TMouseSettings** popisuje univerzální nastavení myši (přip. dotykového panelu). Tuto strukturu využívají metody **GetSettings** a **SetSettings** třídy **TMouseDriver** (viz. kapitoly 7.3.5.31, 7.3.5.31).

```
  TMouseSettings = record
    Size          : Word;
    CheckSum      : Word;
  { Standardni nastaveni mysi }
    DblClickDelay : Integer;
    DblClickArea  : Integer;
    RepeatDelay   : Integer;
    RepeatRate    : Integer;
  { Nastaveni radice touchpanelu }
    PressDelay    : Byte;
    ReleaseDelay  : Byte;
    FilterWeight  : Byte;
    Stabilization : Byte;
    Deglitch      : Byte;
    ChargeWS      : Byte;
  __Reserved      : array[0..3] of Byte;
  { Kalibracni konstanty }
    Calibration   : TMouseCalibration;
  end;
```

Význam jednotlivých položek je uveden v následující tabulce:

Položka	Popis
Size	Položka Size obsahuje délku struktury, tj. SizeOf(**TMouseSettings**).
CheckSum	Položka CheckSum obsahuje kontrolní součet celé struktury. Kontrolní součet je počítán pomocí funkce **GetCheckSum**

	(viz. kapitola 7.4.3).
DblClickDelay	Položka DblClick area udává maximální čas mezi dvěma stisky tlačítka myši, které je ještě detekováno jako dvojklik. Hodnota je zadávána v milisekundách.
DblClickArea	Položka DblClickArea udává rozptyl souřadnic při dvojitém stisku tlačítka myši, které je ještě detekováno jako dvojklik. Hodnota je udávaná v pixelech.
RepeatDelay	Položka RepeatDelay určuje zpoždění před automatickým opakováním stisku tlačítka myši nebo dotykového panelu. Hodnota je zadávána v milisekundách.
RepeatRate	Položka RepeatRate určuje periodu automatického opakování stisku tlačítka myši nebo dotykového panelu. Hodnota je zadávaná v milisekundách.
PressDelay ReleaseDelay FilterWeight Stabilization Deglitch ChargeWS	Tyto položky obsahují nastavení řadiče dotykového panelu. Viz. struktura **TTCDSettings** v dokumentaci ke knihovně TPDrv.
Calibration	Nastavení konstant pro převod souřadnic dotykového panelu na souřadnice displeje (viz. kapitola 7.2.12)

7.2.14. TDisplaySettings

Struktura **TDisplaySettings** popisuje univerzální nastavení displeje. Tuto strukturu využívají metody **GetSettings** a **SetSettings** třídy **TDisplayDriver** (viz. kapitoly 7.3.7.24, 7.3.7.25);

```
TDisplaySettings = record
  Size              : Word;     { Velikost struktury          }
  CheckSum          : Word;     { Kontrolni soucet struktury  }
  Contrast          : Integer;  { Kontrast displeje           }
  Brightness        : Integer;  { Jas displeje                }
  AutoOffDelay      : Integer;  { Automaticke vypnuti po [s]  }
  AutoOffBrightness : Integer;  { Jas ve vypnutem stavu       }
end;
```

Význam jednotlivých položek je uveden v následující tabulce:

Položka	**Popis**
Size	Položka Size obsahuje délku struktury, tj. SizeOf(TDisplaySettings).
CheckSum	Položka CheckSum obsahuje kontrolní součet celé struktury. Kontrolní součet je počítán pomocí funkce **GetCheckSum** (viz. kapitola 7.4.3).
Contrast	Položka Contrast určuje aktuální úroveň kontrastu displeje. Hodnota se může pohybovat v rozsahu –255 až 255.
Brightness	Položka Brightness určuje aktuální úroveň jasu (podsvícení) displeje. Hodnota se může pohybovat v rozsahu 0 až 16. Při

	nulové hodnotě je jas snížen na nejnižší možnou mez (podsvícení je vypnuto).
AutoOffDelay	Položka AutoOffDelay obsahuje čas, po kterém je hodnota jasu displeje snížena na úroveň zadanou položkou AutoOffBrightness v případě, že ze vstupních zařízení (myši a klávesnice) nepřichází žádné události. Hodnota je zadávána v sekundách.
AutoOffBrightness	Položka RepeatRate určuje periodu automatického opakování stisku tlačítka myši nebo dotykového panelu. Hodnota se může pohybovat v rozsahu 0 až 16, stejně jako položka Brightness.

7.3. Třídy

7.3.1. Třída TTimerEx

Třída TTimerEx slouží k odměřování časového intervalu. Třída neobsahuje konstruktor ani destruktor, jedná se pouze o zapouzdření jedné položky a několika metod.

```
PTimerEx = ^TTimerEx;
TTimerEx = object
public
  TickCount : Longint;

  procedure SetTime( AAddend: Longint );
  procedure ShiftTime( AAddend: Longint );
  function  SoonerThan( const ATimer: TTimerEx ): Boolean;
  function  LaterThan( const ATimer: TTimerEx ): Boolean;
  function  Expired: Boolean;
  function  TimeLeft: Longint;
end;
```

*Pozn: Třída **TTimerEx** používá pro odměřování času počet přerušení IRQ0 (tj. globální proměnnou na adrese $0040:$006C spravovanou BIOSem. Tato proměnná je inkrementována s každým přerušení IRQ0, tj. cca každých 55ms. Z tohoto faktu se odvíjí přesnost časovače a minimální nastavitelný časový interval.*

7.3.1.1. Položka TTimerEx.TickCount

Položka **TickCount** obsahuje nastavený okamžik vypršení časovače. Je určena pouze ke čtení, veškeré modifikace položky lze provádět pouze pomocí metod **SetTime** a **ShiftTime**.

```
TickCount : Longint;
```

7.3.1.2. Metoda TTimerEx.SetTime

Metoda **SetTime** nastaví okamžik vypršení časovače na aktuální čas zvýšený o zadaný počet milisekund.

```
procedure SetTime( AAddend: Longint );
```

Parametry:

AAddend	Počet milisekund přičítaných k aktuálnímu času. Horní limit parametru je 12 hodin.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Metoda **SetTime** zjistí aktuální čas, přičte zadaný počet milisekund a výsledek uloží do položky **TickCount**.

7.3.1.3. Metoda TTimerEx.ShiftTime

Metoda **ShiftTime** upraví okamžik vypršení časovače tak, že k aktuální hodnotu zvýší o zadaný počet milisekund.

```
procedure ShiftTime( AAddend: Longint );
```

Parametry:

AAddend	Počet milisekund přičítaných k okamžiku vypršení časovače.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Metoda **ShiftTime** převede zadaný počet milisekund na počet přerušení IRQ0 a přičte je k položce **TickCount**. Je potřeba si uvědomit, že při tomto převodu dochází k zaokrouhlování a proto nejmenší možný počet přičtených milisekund je dán periodou přerušení IRQ (tj. cca 55ms).

7.3.1.4. Metoda TTimerEx.SoonerThan

Metoda **SoonerThan** porovnává okamžiky vypršení dvou časovačů.

```
function  SoonerThan( const ATimer: TTimerEx ): Boolean;
```

Parametry:

ATimer	Odkaz na instanci časovače, se kterým se porovnává.

Návratové hodnoty:

Metoda vrací hodnotu True, pokud časovač vyprší dříve než časovač daný parametrem ATimer. V opačném případě vrací hodnotu False.

Poznámky:

Symbolicky lze vyjádřit prováděné porovnávání pomocí výrazu Self.TickCount < ATimer.TickCount. Vzhledem k tomu, že hodnota TickCount je shora a zdola omezená je implementace porovnání poněkud komplikovanější.

7.3.1.5. Metoda TTimerEx.LaterThan

Metoda **LaterThan** porovnává okamžiky vypršení dvou časovačů.

```
function  LaterThan( const ATimer: TTimerEx ): Boolean;
```

Parametry:

ATimer	Odkaz na instanci časovače, se kterým se porovnává.

Návratové hodnoty:

Metoda vrací hodnotu True, pokud časovač vyprší ve stejný okamžik nebo později než časovač daný parametrem ATimer. V opačném případě vrací hodnotu False.

Poznámky:

Symbolicky lze vyjádřit prováděné porovnávání pomocí výrazu Self.TickCount >= ATimer.TickCount. Vzhledem k tomu, že hodnota TickCount je shora a zdola omezená je implementace porovnání poněkud komplikovanější.

7.3.1.6. Metoda TTimerEx.Expired

Metoda **Expired** zjišťuje, zda nastavený časový interval uplynul.

```
function  Expired: Boolean;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda vrací hodnotu True, pokud nastavený časovač uplynul. V opačném případě vrací hodnotu False.

Poznámky:

7.3.1.7. Metoda TTimerEx.TimeLeft

Metoda **TimeLeft** zjišťuje počet milisekund zbývajících do uplynutí nastaveného časového intervalu.

```
function  TimeLeft: Longint;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda vrací počet milisekund zbývajících do uplynutí nastaveného časového intervalu. Pokud v okamžiku volání časový interval již uplynul, pak návratová je hodnota záporná.

Poznámky:

Přesnost této metody je dána periodou přerušení IRQ0 (tj. cca 55ms).

7.3.2. Třída TEventQueue

Třída **TEventQueue** slouží jako fronta událostí (FIFO) pro různé účely. Mohou ji využít ovladače vstupních zařízení a využívá ji také třída **TInputDriver** pro implementaci fronty uživatelských událostí.

```
PEventQueue = ^TEventQueue;
TEventQueue = object( TObject )
public
  constructor Init( AMaxEvents: Integer );
  destructor Done; virtual;
  procedure Clear;
  function  InsertEvent( const AEvent: TEvent ): Boolean;
  function  RemoveEvent( var AEvent: TEvent ): Boolean;
end;
```

Fronta událostí je implementovaná jako kruhový buffer o neměnné velikosti určené parametrem konstruktoru. Třída nabízí pouze tři metody pro vymazání všech položek z fronty, uložení položky na konec fronty a vyjmutí položky z čela fronty.

7.3.2.1. Konstruktor TEventQueue.Init

Konstruktor **Init** provede inicializaci instance třídy.

```
constructor  Init( AMaxEvents: Integer ): Boolean;
```

Parametry:

AMaxEvent	Maximální počet položek ve frontě.

Návratové hodnoty:

Konstruktor nevrací žádnou hodnotu.

Poznámky:

Konstruktor inicializuje instanci a na hromadě vytvoří pole položek TEvent, jehož délka je dána parametrem konstruktoru.

7.3.2.2. Destruktor TEventQueue.Done

Destruktor **Done** provede uvolnění prostředků alokovaných třídou.

```
destructor  Done; virtual;
```

Parametry:

Destruktor nemá žádné parametry.

Návratové hodnoty:

Destruktor nevrací žádnou hodnotu.

Poznámky:

Destruktor uvolní z hromady pole položek alokovaných v konstruktoru.

7.3.2.3. Metoda TEventQueue.Clear

Metoda **Clear** odstraní z fronty všechny položky. Po jejím provedení bude tedy fronta prázdná.

```
procedure  Clear;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

7.3.2.4. Metoda TEventQueue.InsertEvent

Metoda **InsertEvent** provede vložení struktury události na konec fronty.

```
function  InsertEvent( const AEvent: TEvent ): Boolean;
```

Parametry:

AEvent	Vkládaná událost.

Návratové hodnoty:

Metoda vrací hodnotu True, pokud se událost podařilo do fronty vložit. V opačném případě (fronta je plná) událost do fronty vložena nebyla.

7.3.2.5. Metoda TEventQueue.RemoveEvent

Metoda **RemoveEvent** provede vyjmutí události z čela fronty.

```
function  RemoveEvent( var AEvent: TEvent ): Boolean;
```

Parametry:

AEvent	Parametr, do kterého je uložena vyjmutá událost..

Návratové hodnoty:

Metoda vrací hodnotu True, pokud se událost podařilo z fronty vyjmout. V opačném případě (fronta je prázdná) žádná událost z fronty vyjmuta nebyla a parametr AEvent má nedefinovanou hodnotu.

7.3.3. Třída TInputDriver

```
PInputDriver = ^TInputDriver;
TInputDriver = object( TObject )
public
  Keyboard    : PKeyboardDriver;   { Zarizeni klavesnice          }
  Mouse       : PMouseDriver;      { Polohovaci zarizeni          }
  Timer       : PTimerDriver;      { Casovac                      }
  UserQueue   : TEventQueue;       { Fronta uzivatelskych udalosti}

  constructor Init( AKeyboard: PKeyboardDriver;
                    AMouse: PMouseDriver );
  destructor  Done; virtual;

  function  Initialize: Boolean;
  procedure Finalize;
  procedure Tick; virtual;
  function  Broadcast( ACommand: Word; AParam: Pointer ): Boolean;
  function  PutEvent( AEvent: TEvent ): Boolean;
  procedure GetEvent( var AEvent: TEvent; ANoTimer: Boolean );
end;
```

7.3.3.1. Položka TInputDriver.Keyboard

Položka **Keyboard** obsahuje odkaz na instanci potomka třídy **TKeyboardDriver** (viz. kapitola 7.3.4). V případě, že terminál není vybaven klávesnicí, může tato položka obsahovat hodnotu **nil**. Položka je určena pouze ke čtení, její hodnota se nastavuje při volání konstruktoru.

```
Keyboard : PKeyboardDriver;
```

7.3.3.2. Položka TInputDriver.Mouse

Položka **Mouse** obsahuje odkaz na instanci potomka třídy **TMouseDriver** (viz. kapitola 7.3.4). V případě, že terminál není vybaven myší nebo dotykovým panelem, může tato položka obsahovat hodnotu **nil**. Položka je určena pouze ke čtení, její hodnota se nastavuje při volání konstruktoru.

```
Mouse : PMouseDriver;
```

7.3.3.3. Položka TInputDriver.Timer

Položka **Timer** obsahuje odkaz na instanci třídy **TTimerDriver** (viz. kapitola 7.3.6). Položka je určena pouze ke čtení, její hodnota je automaticky nastavena při volání konstruktoru.

```
Timer : PTimerDriver;
```

7.3.3.4. Položka TInputDriver.UserQueue

Položka **UserQueue** obsahuje odkaz na instanci třídy **TEventQueue** (viz. kapitola 7.3.2). Položka je určena pouze ke čtení, její hodnota je automaticky nastavena při volání konstruktoru.

```
UserQueue : PEventQueue;
```

7.3.3.5. Konstruktor TInputDriver.Init

Konstruktor **Init** provede inicializaci instance třídy.

```
constructor Init( AKeyboard: PKeyboardDriver; AMouse: PMouseDriver );
```

Parametry:

AKeyboard	Odkaz na vytvořenou instanci potomka třídy **TKeyboardDriver** nebo hodnota **nil**.
AMouse	Odkaz na vytvořenou instanci potomka třídy **TMouseDriver** nebo hodnota **nil**.

Návratové hodnoty:

Konstruktor nevrací žádnou hodnotu.

Poznámky:

Konstruktor inicializuje instanci a na hromadě vytvoří instanci třídy **TTimerDriver**, **TEventQueue** a odkazy na ně uloží do položek **Timer** a **UserQueue**. Dále inicializuje položky **Keyboard** a **Mouse** podle parametrů konstruktoru.

7.3.3.6. Destruktor TInputDriver.Done

Destruktor **Done** provede uvolnění prostředků alokovaných instancí.

```
destructor Done; virtual;
```

Parametry:

Destruktor nemá žádné parametry.

Návratové hodnoty:

Destruktor nevrací žádnou hodnotu.

Poznámky:

Destruktor uvolní z paměti instance odkazované v položkách **Keyboard**, **Mouse**, **Timer** a **UserQueue**.

7.3.3.7. Metoda TInputDriver.Initialize

Metoda **Initialize** provádí inicializaci ovladače vstupů a všech vložených ovladačů, tj. klávesnice a myši.

```
function Initialize: Boolean;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda vrací hodnotu True pokud inicializace ovladače vstupů a všech vložených ovladačů proběhla úspěšně. V opačném případě vrací hodnotu False.

Poznámky:

Metodu **Initialize** je nutné volat poté co byla instance vytvořena voláním konstruktoru. Bez jejího volání nebudou vstupní zařízení pracovat správně příp. nebudou pracovat vůbec.
Metoda **Initialize** interně volá metody **Initialize** ovladače myši a klávesnice (instance odkazované položkami **Mouse** a **Keyboard**). Pokud alespoň jedna z nich vrátí hodnotu False, vrátí ji i tato metoda. Pokud metoda **Initialize** neproběhla úspěšně, je možné, že některé z ovladačů již přesto inicializovány byly. Proto je potřeba pro správné uvolnění prostředků před volání destruktoru zavolat metodu **Finalize**.

7.3.3.8. Metoda TInputDriver.Finalize

Metoda **Finalize** provádí uvolnění prostředků alokovaných ovladače vstupů a vložených ovladačů, tj. ovladačů klávesnice a myši.

```
function Finalize;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Metodu **Finalize** je nutné volat před voláním destruktoru a úplným zrušením instance.
Metoda **Finalize** interně volá metody **Finalize** ovladače myši a klávesnice (instance odkazované položkami **Mouse** a **Keyboard**).

7.3.3.9. Metoda TInputDriver.Tick

Metoda **Tick** provádí jeden krok automatu vstupních ovladačů.

```
procedure Tick; virtual;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Metodu **Tick** interně volá metody Tick ovladače klávesnice a myši (dotykového panelu). Metodu **Tick** je nutné volat v pravidelných intervalech. Obvykle je metoda **Tick** periodicky volána z hlavní smyčky vizulizačního systému.

7.3.3.10. Metoda TInputDriver.Broadcast

Metoda **Broadcast** vloží do uživatelské fronty událost typu oběžník

```
function Broadcast( ACommand: Word; AParam: Pointer ): Boolean;
```

Parametry:

ACommand	Parametr ACommand obsahuje tzv. příkaz oběžníku, tj. rozlišení typu oběžníku. Tato hodnota bude před vložením struktury události TEvent do fronty uložena do její položky Command. Obvykle se jedná o jednu z konstant s prefixem cm_, které nejsou definovány touto knihovnou.
AParam	Pomocný (uživatelský) ukazatel ukládaný do položky Param struktury TEvent. Význam tohoto parametru je dán typem oběžníku.

Návratové hodnoty:

Metoda **Broadcast** vrací hodnotu True, pokud se podařilo vytvořit událost typu oběžník a vložit ji do uživatelské fronty. V případě, že je fronta plná metoda vrátí hodnotu False.

Poznámky:

Po úspěšném provedení metody je událost oběžníku (evBroadcast) vložena do uživatelské fronty. Tato událost bude zpracována s vyšší prioritou než ostatní typy zpráv (klávesnice, myši, časovače).

7.3.3.11. Metoda TInputDriver.PutEvent

Metoda **PutEvent** vloží do uživatelské fronty událost.

```
function  PutEvent( AEvent: TEvent ): Boolean;
```

Parametry:

AEvent	Parametr AEvent obsahuje vyplněnou událost, která se po provedení metody vloží do uživatelské fronty.

Návratové hodnoty:

Metoda **PutEvent** vrací hodnotu True, pokud se událost vložit do uživatelské fronty. V případě, že je fronta plná metoda vrátí hodnotu False.

Poznámky:

Po úspěšném provedení metody je událost vložena do uživatelské fronty. Tato událost bude zpracována s vyšší prioritou než ostatní typy zpráv (klávesnice, myši, časovače).

7.3.3.12. Metoda TInputDriver.GetEvent

Metoda **GetEvent** vloží do uživatelské fronty událost.

```
procedure GetEvent( var AEvent: TEvent; ANoTimer: Boolean );
```

Parametry:

AEvent	Struktura daná parametrem AEvent bude po úspěšném provedení funkce obsahovat nejstarší událost (dle priority). V případě, že žádná událost není k dispozici, bude po provedení funkce položka Code struktura AEvent nastavena na hodnotu evNothing.
ANoTimer	Pokud je parametr ANoTimer nastaven na hodnotu False, nebude při volání metody **GetEvent** vyjmuta žádná zpráva týkající se vypršení limitu časovače.

Návratové hodnoty:

Metoda **GetEvent** nevrací žádnou hodnotu.

Poznámky:

Metoda **GetEvent** postupně testuje jednotlivé vstupy podle priority.

Priorita	Typ události
Nejvyšší	Uživatelské události
	Události klávesnice
	Události myši (dotykového panelu)
Nejnižší	Událost vypršení limitu časovače

7.3.4. Třída TKeyboardDriver

Abstraktní třída **TKeybardDriver** definuje obecné rozhraní ovladačů klávesnice.

```
PKeyboardDriver = ^TKeyboardDriver;
TKeyboardDriver = object( TObject )
public
  function  Initialize: Boolean; virtual;
  procedure Finalize; virtual;
  procedure Tick; virtual;
  procedure GetEvent( var Event: TEvent ); virtual;
  procedure SetRepeatDelay( AValue: Integer ); virtual;
  function  GetRepeatDelay: Integer; virtual;
  procedure SetRepeatRate( AValue: Integer ); virtual;
  function  GetRepeatRate: Integer; virtual;

  function  SetSettings( const ASettings: TKeyboardSettings ):
              Boolean; virtual;
  procedure GetSettings( var ASettings: TKeyboardSettings );
              virtual;
end;
```

Třída **TKeyboardDriver** definuje metody pro nastavování různých parametrů ovladače klávesnice. Potomci této třídy mohou předefinovat pouze ty metody, které mají pro dané zařízení smysl. Každý potomek by musí předefinovat tyto metodu **GetEvent**. Měl by také předefinovat podle potřeby metody **Initialize**, **Finalize** a **Tick**.

7.3.4.1. Metoda TKeyboardDriver.Initialize

Metoda **Initialize** provádí inicializaci ovladače klávesnice.

```
function Initialize: Boolean; virtual;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda vrací hodnotu True pokud inicializace ovladače klávesnice proběhla úspěšně. V opačném případě vrací hodnotu False.

Poznámky:

Metodu **Initialize** je volána poté co byla vytvořena instance třídy voláním konstruktoru. Bez jejího volání nebude ovladač klávesnice pracovat správně příp. nebude pracovat vůbec.

Přímo ve třídě **TKeyboardDriver** je metoda **Initialize** definovaná jako prázdná a vrací implicitně hodnotu True. Potomci třídy mohou tuto metodu předefinovat.

7.3.4.2. Metoda TKeyboardDriver.Finalize

Metoda **Finalize** provádí uvolnění prostředků alokovaných ovladačem klávesnice.

```
procedure Finalize; virtual;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Metodu **Finalize** je volána před voláním destruktoru třídy **TKeyboardDriver**.

Přímo ve třídě **TKeyboardDriver** je metoda **Finalize** definovaná jako prázdná. Potomci třídy mohou tuto metodu předefinovat.

7.3.4.3. Metoda TKeyboardDriver.Tick

Metoda **Tick** provádí jeden krok automatu ovladače klávesnice.

```
procedure Tick; virtual;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Metodu **Tick** je nutné volat v pravidelných intervalech. Obvykle je metoda **Tick** periodicky volána z hlavní smyčky vizulizačního systému.

Přímo ve třídě **TKeyboardDriver** je metoda **Tick** definovaná jako prázdná. Potomci třídy mohou tuto metodu předefinovat.

7.3.4.4. Metoda TKeyboardDriver.GetEvent

Metoda **GetEvent** předá nejstarší událost a odstraní ji z fronty událostí ovladače klávesnice.

```
procedure GetEvent( var AEvent: TEvent ); virtual;
```

Parametry:

AEvent	Po provedení metody je do parametru AEvent uložena událost typu evKeyDown a jsou vyplněny položky KeyCode,

CharCode. Položka VirtKey je nastavena na 0. V případě, že nebyla stisknuta žádná klávesa je vyplněna pouze položka Code hodnotou evNothing.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Přímo ve třídě **TKeyboardDriver** je metoda **GetEvent** definovaná jako abstraktní. Potomci třídy musí tuto metodu předefinovat.

7.3.4.5. Metoda TKeybaordDriver.SetRepeatDelay

Metoda **SetRepeatDelay** nastavuje zpoždění před automatickým opakováním kláves, tj. dobu držení klávesy, než se začne automaticky opakovat událost stisku.

```
procedure SetRepeatDelay( AValue: Integer ); virtual;
```

Parametry:

AValue	Zpoždění zadávané v milisekundách. Hodnota se může pohybovat rozsahu 0 až 32767. Pokud je zadána hodnota 0, je automatické opakování kláves vypnuto.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Metoda **SetRepeatDelay** je ve třídě **TKeyboardDriver** definovaná jako prázdná. Potomci třídy mohou tuto metodu předefinovat, pokud obsluhovaná klávesnice umožňuje nastavovat tento parametr.

7.3.4.6. Metoda TKeyboardDriver.GetRepeatDelay

Metoda **GetRepeatDelay** vrací zpoždění před automatickým opakováním kláves, tj. dobu držení klávesy, než se začne automaticky opakovat událost stisku.

```
function  GetRepeatDelay: Integer; virtual;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda vrací čas v milisekundách.

Poznámky:

Metoda **GetRepeatDelay** je ve třídě **TKeyboardDriver** definovaná, tak že vždy vrací hodnotu 0. Potomci třídy mohou tuto metodu předefinovat, pokud obsluhovaná klávesnice umožňuje nastavovat tento parametr.

7.3.4.7. Metoda TKeyboardDriver.SetRepeatRate

Metoda **SetRepeatRate** nastavuje periodu automatického opakování kláves, tj. prodlevu mezi dvěma automaticky vygenerovanými událostí stisku klávesy.

```
procedure SetRepeatRate( AValue: Integer ); virtual;
```

Parametry:

AValue	Perioda zadávaná v milisekundách. Hodnota se může pohybovat rozsahu 1 až 32767. Pokud je zadána hodnota 0, je automatické opakování kláves vypnuto.

Návratové hodnoty:

Metoda vrací čas v milisekundách.

Poznámky:

Metoda **SetRepeatDelay** je ve třídě **TKeyboardDriver** definovaná jako prázdná. Potomci třídy mohou tuto metodu předefinovat, pokud obsluhovaná klávesnice umožňuje nastavovat tento parametr.

7.3.4.8. Metoda TKeyboardDriver.GetRepeatRate

Metoda **SetRepeatRate** vrací periodu automatického opakování kláves, tj. prodlevu mezi dvěma automaticky vygenerovanými událostí stisku klávesy.

```
function  GetRepeatRate: Integer; virtual;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda vrací čas v milisekundách.

Poznámky:

Metoda **GetRepeatRate** je ve třídě **TKeyboardDriver** definovaná, tak že vždy vrací hodnotu 0. Potomci třídy mohou tuto metodu předefinovat, pokud obsluhovaná klávesnice umožňuje nastavovat tento parametr.

7.3.4.9. Metoda TKeyboardDriver.SetSettings

Metoda **SetSettings** nastavuje veškeré parametry klávesnice z připravené struktury (viz. kapitola 7.2.11).

```
function  SetSettings( const ASettings: TKeyboardSettings ):
            Boolean; virtual;
```

Parametry:

ASettings	Parametr ASettings obsahuje nastavení parametrů ovladače klávesnice. Všechny položky struktury musí být vyplněny.

Návratové hodnoty:

Pokud metoda **SetSettings** proběhla úspěšně, vrací hodnotu True. V opačném případě vrací hodnotu False..

Poznámky:

Metoda **SetSetttings** vyžaduje, aby byly vyplněny všechny položky struktury ASettings (viz. kapitola 7.2.11), včetně kontrolního součtu. Pokud kontrolní součet nesouhlasí metoda vrátí hodnotu False a v tomto případě není změněn žádný parametr ovladače klávesnice. Metoda interně volá metody **SetRepeatDelay** a **SetRepeatRate**.

7.3.4.10. Metoda TKeyboardDriver.GetSettings

Metoda **GetSettings** uloží do připravené struktury (viz. kapitola 7.2.11) veškeré nastavení ovladače klávesnice.

```
procedure GetSettings( var ASettings: TKeyboardSettings ); virtual;
```

Parametry:

ASettings	Po provedení metody bude parametr ASettings vyplněn aktuálním nastavení ovladače klávesnice.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Po provedení metody bude parametr ASettings vyplněn aktuálním nastavení ovladače klávesnice. Bude proveden výpočet kontrolního součtu strukturu pomocí funkce **GetChecksum** a tento kontrolní součet bude uložen do položky CheckSum struktury spolu s nastavením ovladače. Metoda interně volá metody **GetRepeatDelay** a **GetRepeatRate**.

7.3.5. Třída TMouseDriver

Abstraktní třída **TMouseDriver** definuje obecné rozhraní ovladačů polohovacích zařízení (myši, dotykového panelu).

```
PMouseDriver = ^TMouseDriver;
TMouseDriver = object( TObject )
public
  DispDriver : PDisplayDriver;

  function  Initialize: Boolean; virtual;
  procedure Finalize; virtual;
  procedure Tick; virtual;
  procedure GetEvent( var Event: TEvent ); virtual;
  procedure SetDisplayDriver( ADisplayDriver: PDisplayDriver );
  procedure SetMode( ACalibration: Boolean ); virtual;
  function  Calibrate( DispA, MouseA, DispB, MouseB,
              DispC, MouseC : TPoint ): Boolean; virtual;

  procedure SetPressDelay( AValue: Byte ); virtual;
  function  GetPressDelay: Byte; virtual;
  procedure SetReleaseDelay( AValue: Byte ); virtual;
  function  GetReleaseDelay: Byte; virtual;
  procedure SetDeglitch( AValue: Byte ); virtual;
  function  GetDeglitch: Byte; virtual;
  procedure SetChargeWS( AValue: Byte ); virtual;
  function  GetChargeWS: Byte; virtual;
  procedure SetFilterWeight( AValue: Byte ); virtual;
  function  GetFilterWeight: Byte; virtual;
  procedure SetStabilization( AValue: Byte ); virtual;
  function  GetStabilization: Byte; virtual;
  procedure SetDblClickDelay( AValue: Integer ); virtual;
  function  GetDblClickDelay: Integer; virtual;

  procedure SetDblClickArea( AValue: Integer ); virtual;
  function  GetDblClickArea: Integer; virtual;
  procedure SetRepeatDelay( AValue: Integer ); virtual;
  function  GetRepeatDelay: Integer; virtual;
  procedure SetRepeatRate( AValue: Integer ); virtual;
  function  GetRepeatRate: Integer; virtual;
  procedure GetCalibration( var ACalibration:
              TMouseCalibration ); virtual;
  procedure SetCalibration( const ACalibration:
              TMouseCalibration ); virtual;
  procedure GetSettings( var ASettings: TMouseSettings ); virtual;
  function  SetSettings( const ASettings: TMouseSettings ):
              Boolean; virtual;
end;
```

Ovladač myši se může nacházet v dvou režimech: v režimu *provozním* a *kalibračním*. V kalibračním režimu nedochází ke konverzi souřadnic vstupního zařízení na souřadnice displeje. Režim se nastavuje pomocí metody **SetMode** (viz. kapitola 7.3.5.7). Pro myš a podobná zařízení nemá kalibrační režim smysl (ovladač myši se chová v provozním i v kalibračním režimu identicky). Kalibrační režim je však nezbytný např. pro dotykový panel. Samotná kalibrace se provádí pomocí metody **Calibrate** (viz. kapitola 7.3.5.8).

Třída **TMouseDriver** definuje metody pro nastavování různých parametrů ovladače

polohovacího zařízení. Potomci této třídy mohou předefinovat pouze ty metody, které mají pro dané zařízení smysl. Každý potomek by musí předefinovat tyto metodu **GetEvent**. Měl by také předefinovat podle potřeby metody **Initialize**, **Finalize** a **Tick**. Pokud dané zařízení vyžaduje kalibraci, musí předefinovat také metody **SetMode**, **Calibrate**, **GetCalibration**, **SetCalibration**.

7.3.5.1. Položka TMouseDriver.DispDriver

Položka **DispDriver** se odkazuje na ovladač displeje a slouží pro účely vykreslování ukazatele myši. Pokud ukazatel myši nemá být vidět, může být tato položka nastavena na hodnotu **nil**. Položka **DispDriver** je určena pouze pro čtení a její hodnota se nastavuje pomocí metody **SetDisplayDriver** (viz. kapitola 7.3.5.6).

```
DispDriver : PDisplayDriver;
```

7.3.5.2. Metoda TMouseDriver.Initialize

Metoda **Initialize** provádí inicializaci ovladače myši.

```
function Initialize: Boolean; virtual;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda vrací hodnotu True pokud inicializace ovladače myši proběhla úspěšně. V opačném případě vrací hodnotu False.

Poznámky:

Metodu **Initialize** je volána poté co byla vytvořena instance třídy voláním konstruktoru. Bez jejího volání nebude ovladač myši pracovat správně příp. nebude pracovat vůbec.

Přímo ve třídě **TMouseDriver** je metoda **Initialize** definovaná jako prázdná a vrací implicitně hodnotu True. Potomci třídy mohou tuto metodu předefinovat.

7.3.5.3. Metoda TMouseDriver.Finalize

Metoda **Finalize** provádí uvolnění prostředků alokovaných ovladačem myši.

```
procedure Finalize; virtual;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Metodu **Finalize** je volána před voláním destruktoru třídy **TMouseDriver**.

Přímo ve třídě **TMouseDriver** je metoda **Finalize** definovaná jako prázdná. Potomci třídy mohou tuto metodu předefinovat.

7.3.5.4. Metoda TMouseDriver.Tick

Metoda **Tick** provádí jeden krok automatu ovladače myši.

```
procedure Tick; virtual;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Metodu **Tick** je nutné volat v pravidelných intervalech. Obvykle je metoda **Tick** periodicky volána z hlavní smyčky vizulizačního systému.

Přímo ve třídě **TMouseDriver** je metoda **Tick** definovaná jako prázdná. Potomci třídy mohou tuto metodu předefinovat.

7.3.5.5. Metoda TMouseDriver.GetEvent

Metoda **GetEvent** předá nejstarší událost a odstraní ji z fronty událostí ovladače myši.

```
procedure GetEvent( var AEvent: TEvent ); virtual;
```

Parametry:

AEvent	Po provedení metody je do parametru AEvent uložena událost typu evMouseXXX a jsou vyplněny položky Buttons a Pos. V případě, že ve fronta událostí ovladače myši je prázdná, pak je vyplněna pouze položka Code hodnotou evNothing.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Ovladač myši může generovat následující uvedené v tabulce níže. U všech typů událostí musí být vyplněny položky Pos a Buttons struktury **TEvent**.

Událost	Událost je povinná	Popis události
evMouseDown	Ano	Stisk tlačítka myši (stisk dotykového panelu)
evMouseUp	Ano	Uvolnění tlačítka myši (uvolnění dotykového panelu)
evMouseMove	Ano	Pohyb ukazatele myši (tlačítka mohou být stisknuta i uvolněna)
evMouseDbl	Ne	Dvojklik tlačítka myši („dvojstisk“ dotykového panelu)
evMouseRep	Ne	Automatické opakování stisku tlačítka

Přímo ve třídě **TMouseDriver** je metoda **GetEvent** definovaná jako abstraktní. Potomci třídy musí tuto metodu předefinovat.

7.3.5.6. Metoda TMouseDriver.SetDisplayDriver

Metoda **SetDisplayDriver** nastavuje položku instance DispDriver (viz. kapitola 7.3.5.1).

```
procedure SetDisplayDriver( ADisplayDriver: PDisplayDriver );
```

Parametry:

ADisplayDriver Odkaz na instanci ovladače displeje.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

7.3.5.7. Metoda TMouseDriver.SetMode

Metoda **SetMode** nastavuje režim ovladače myši.

```
procedure SetMode( ACalibration: Boolean ); virtual;
```

Parametry:

ACalibration Pokud je tento parametr nastaven na True, bude se ovládač myši nacházet po návratu metody v kalibračním režimu. V opakčné případě dojde k přepnutí do režimu provozního.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Ovladač myši se může nacházet v dvou režimech: v režimu *provozním* a *kalibračním*. V kalibračním režimu nedochází ke konverzi souřadnic vstupního zařízení na souřadnice displeje. Pro myš a podobná zařízení nemá kalibrační režim smysl (ovladač myši se chová v provozním i v kalibračním režimu identicky). Kalibrační režim je však nezbytný např. pro dotykový panel. Samotná kalibrace se provádí pomocí metody **Calibrate** (viz. kapitola 7.3.5.8).

7.3.5.8. Metoda TMouseDriver.Calibrate

Metoda **Calibrate** provádí kalibraci převodních konstant ze souřadnic polohovacího zařízení na souřadnice displeje.

```
function  Calibrate( DispA, MouseA, DispB, MouseB,
            DispC, MouseC : TPoint ): Boolean; virtual;
```

Parametry:

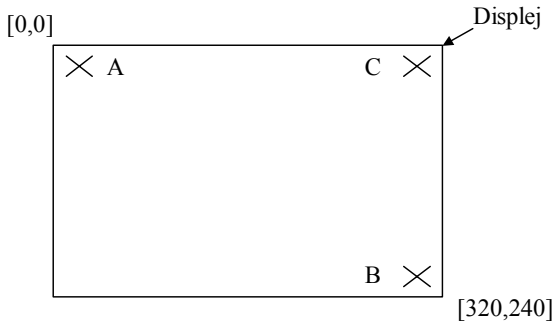
DispA	Souřadnice displeje bodu A.
MouseA	Souřadnice polohovacího zařízení bodu A.
DispB	Souřadnice displeje bodu B.
MouseB	Souřadnice polohovacího zařízení bodu B.
DispC	Souřadnice displeje bodu C.
MouseC	Souřadnice polohovacího zařízení bodu C.

Návratové hodnoty:

Metoda vrací hodnotu True, pokud byla kalibrace úspěšná. Pokud metoda vrátí hodnotu False, kalibrace neproběhla a nastavení kalibračních konstant nebylo změněno.

Poznámky:

Metoda **Calibrate** umožňuje tříbodovou kalibraci, tj. kalibraci pomocí třech bodů umístěných obvykle v rozích displeje. Viz. následující obrázek:



Uživatel nastaví v kalibračním režimu postupně označí body A, B a C. Poté se zavolá

metoda **Calibrate** s odměřenými souřadnicemi (parametry MouseA, MouseB a MouseC) a s požadovanými souřadnicemi (DispA, DispB, DispC). Metoda nastaví kalibrační konstanty (viz. struktura **TMouseCalibration** v kapitole 7.2.12). Pro přímé nastavení a čtení aktuálních kalibračních konstant lze použít metodu **SetCalibration** a **GetCalibration** (viz. kapitoly 7.3.5.29 a 7.3.5.30)

Přímo ve třídě **TMouseDriver** je metoda **Calibrate** definovaná jako prázdná a vrací hodnotu True. Potomci třídy mohou tuto metodu předefinovat.

7.3.5.9. Metoda TTouchPanelDriver.SetPressDelay

Metoda **SetPressDelay** slouží pro nastavení položky PressDelay struktury **TTCDSettings** (viz. kapitola dokumentace ke knihovně TPDrv).

```
procedure SetPressDelay( AValue: Byte ); virtual;
```

Parametry:

AValue	Hodnota parametru.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Význam parametru a jeho meze jsou poplatné konkrétnímu ovladači dotykového panelu (viz. např. dokumentace ke knihovně TPDrv).

7.3.5.10. Metoda TTouchPanelDriver.GetPressDelay

Metoda **GetPressDelay** vrací aktuální nastavení položky PressDelay struktury **TTCDSettings** (viz.dokumentace ke knihovně TPDrv).

```
function GetPressDelay: Byte; virtual;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda vrací aktuální hodnotu parametru. Rozsah návratových hodnot je poplatný konkrétnímu ovladači dotykového panelu (viz. např. dokumentace ke knihovně TPDrv).

7.3.5.11. Metoda TTouchPanelDriver.SetReleaseDelay

Metoda **SetReleaseDelay** slouží pro nastavení položky PressDelay struktury

TTCDSettings (viz.dokumentace ke knihovně TPDrv).

```
procedure SetReleaseDelay( AValue: Byte ); virtual;
```

Parametry:

AValue	Hodnota parametru.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Význam parametru a jeho meze jsou poplatné konkrétnímu ovladači dotykového panelu (viz. např. dokumentace ke knihovně TPDrv).

7.3.5.12. Metoda TTouchPanelDriver.GetReleaseDelay

Metoda **GetReleaseDelay** vrací aktuální nastavení položky ReleaseDelay struktury **TTCDSettings**(viz.dokumentace ke knihovně TPDrv).

```
function GetReleaseDelay: Byte; virtual;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda vrací aktuální hodnotu parametru. Rozsah návratových hodnot je poplatný konkrétnímu ovladači dotykového panelu (viz. např. dokumentace ke knihovně TPDrv).

7.3.5.13. Metoda TTouchPanelDriver.SetDeglitch

Metoda **SetDeglitch** slouží pro nastavení položky Deglitch struktury **TTCDSettings**(viz.dokumentace ke knihovně TPDrv).

```
procedure SetDeglitch( AValue: Byte ); virtual;
```

Parametry:

AValue	Hodnota parametru.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Význam parametru a jeho meze jsou poplatné konkrétnímu ovladači dotykového panelu (viz. např. dokumentace ke knihovně TPDrv).

7.3.5.14. Metoda TTouchPanelDriver.GetDeglitch

Metoda **GetDeglitch** vrací aktuální nastavení položky Deglitch struktury **TTCDSettings** (viz.dokumentace ke knihovně TPDrv).

```
function GetDeglitch: Byte; virtual;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda vrací aktuální hodnotu parametru. Rozsah návratových hodnot je poplatný konkrétnímu ovladači dotykového panelu (viz. např. dokumentace ke knihovně TPDrv).

7.3.5.15. Metoda TTouchPanelDriver.SetChargeWS

Metoda **SetChargeWS** slouží pro nastavení položky ChargeWS struktury **TTCDSettings** (viz.dokumentace ke knihovně TPDrv).

```
procedure SetChargeWS ( AValue: Byte ); virtual;
```

Parametry:

AValue	Hodnota parametru.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Význam parametru a jeho meze jsou poplatné konkrétnímu ovladači dotykového panelu (viz. např. dokumentace ke knihovně TPDrv).

7.3.5.16. Metoda TTouchPanelDriver.GetChargeWS

Metoda **GetChargeWS** vrací aktuální nastavení položky ChargeWS struktury **TTCDSettings** (viz.dokumentace ke knihovně TPDrv).

```
function GetChargeWS: Byte; virtual;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda vrací aktuální hodnotu parametru. Rozsah návratových hodnot je poplatný konkrétnímu ovladači dotykového panelu (viz. např. dokumentace ke knihovně TPDrv).

7.3.5.17. Metoda TTouchPanelDriver.SetFilterWeight

Metoda **SetFilterWeight** slouží pro nastavení položky FilterWeight struktury **TTCDSettings** (viz.dokumentace ke knihovně TPDrv).

```
procedure SetFilterWeight( AValue: Byte ); virtual;
```

Parametry:

AValue	Hodnota parametru.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Význam parametru a jeho meze jsou poplatné konkrétnímu ovladači dotykového panelu (viz. např. dokumentace ke knihovně TPDrv).

7.3.5.18. Metoda TTouchPanelDriver.GetFilterWeight

Metoda **GetFilterWeight** vrací aktuální nastavení položky FilterWeight struktury **TTCDSettings** (viz.dokumentace ke knihovně TPDrv).

```
function GetFilterWeight: Byte; virtual;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda vrací aktuální hodnotu parametru. Rozsah návratových hodnot je poplatný konkrétnímu ovladači dotykového panelu (viz. např. dokumentace ke knihovně TPDrv).

7.3.5.19. Metoda TTouchPanelDriver.SetStabilization

Metoda **SetStabilization** slouží pro nastavení položky Stabilization struktury **TTCDSettings** (viz.dokumentace ke knihovně TPDrv).

```
procedure SetStabilization( AValue: Byte ); virtual;
```

Parametry:

AValue	Hodnota parametru.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Význam parametru a jeho meze jsou poplatné konkrétnímu ovladači dotykového panelu (viz. např. dokumentace ke knihovně TPDrv).

7.3.5.20. Metoda TTouchPanelDriver.GetStabilization

Metoda **GetStabilization** vrací aktuální nastavení položky Stabilization struktury **TTCDSettings** (viz.dokumentace ke knihovně TPDrv).

```
function GetStabilization: Byte; virtual;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda vrací aktuální hodnotu parametru. Rozsah návratových hodnot je poplatný konkrétnímu ovladači dotykového panelu (viz. např. dokumentace ke knihovně TPDrv).

7.3.5.21. Metoda TMouseDriver.SetDblClickDelay

Metoda **SetDblClickDelay** slouží pro nastavení maximálního časového intervalu mezi dvěmi kliknutími detekovanými jako dvojklik.

Parametry:

AValue	Délka časového intervalu v milisekundách.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

7.3.5.22. Metoda TMouseDriver.GetDblClickDelay

Metoda **GetDblClickDelay** vrací hodnotu maximálního časového intervalu mezi dvěmi kliknutími detekovanými jako dvojklik.

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda vrací délku časového intervalu v milisekundách.

Poznámky:

7.3.5.23. Metoda TMouseDriver.SetDblClickArea

Metoda **SetDblClickArea** slouží k nastavení velikosti oblasti dvojkliku, tj. maximální vzdálenosti mezi pozicema dvou kliknutí, detekovanými jako dvojklik.

Parametry:

AValue Počet pixelů.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Optimální hodnota velikosti oblasti dvojkliku závisí na rozměrech bodu displeje. Obvykle se pohybuje mezi 4 až 8 pixely.

7.3.5.24. Metoda TMouseDriver.GetDblClickArea

Metoda **GetDblClickArea** vrací velikost oblasti dvojkliku, tj. maximální vzdálenosti mezi pozicema dvou kliknutí, detekovanými jako dvojklik

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda vrací počet pixelů..

Poznámky:

7.3.5.25. Metoda TMouseDriver.SetRepeatDelay

Metoda **SetRepeatDelay** slouží k nastavení zpoždění generování události evMouseRep při držení tlačítka myši na jednom místě po delší dobu.

Parametry:

AValue Délka časového intervalu v milisekundách.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

7.3.5.26. Metoda TMouseDriver.GetRepeatDelay

Metoda **GetRepeatDelay** vrací nastavené zpoždění generování události evMouseRep při držení tlačítka myši na jednom místě po delší dobu.

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda vrací délku časového intervalu v milisekundách.

Poznámky:

7.3.5.27. Metoda TMouseDriver.SetRepeatRate

Metoda **SetRepeatRate** slouží k nastavení periody generování události evMouseRep při držení tlačítka myši na jednom místě po delší dobu.

Parametry:

AValue Délka časového intervalu v milisekundách.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

7.3.5.28. Metoda TMouseDriver.GetRepeatRate

Metoda **GetRepeatRate** vrací nastavenou periodu generování události evMouseRep při držení tlačítka myši na jednom místě po delší dobu.

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda vrací délku časového intervalu v milisekundách.

Poznámky:

7.3.5.29. Metoda TMouseDriver.SetCalibration

Metoda **SetCalibration** provádí nastavení kalibračních konstant pro převod souřadnic polohovacího zařízení na souřadnice displeje.

```
procedure SetCalibration( const ACalibration:
            TMouseCalibration ); virtual;
```

Parametry:

ACalibration	Parametr odkazující se na strukturu obsahující kalibrační konstanty (viz. kapitola 7.2.12).

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:
Přímo ve třídě **TMouseDriver** je metoda **SetCalibration** definovaná jako prázdná. Pokud potomci třídy obsluhují zařízení vyžadující kalibraci, je potřeba tuto metodu předefinovat.

7.3.5.30. Metoda TMouseDriver.GetCalibration

Metoda **GetCalibration** provádí čtení kalibračních konstant pro převod souřadnic polohovacího zařízení na souřadnice displeje.

```
procedure GetCalibration( var ACalibration:
            TMouseCalibration ); virtual;
```

Parametry:

ACalibration	Parametr bude po návratu metody naplněn kalibračníma konstantama (viz. kapitola 7.2.12).

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Přímo ve třídě **TMouseDriver** je metoda **SetCalibration** definovaná tak, že nastaví položky struktury ACalibration podle následující tabulky.

Položka	Hodnota
XMultiplier	4096
XAddend	0
YMultiplier	4096
YAddend	0
SwapXY	False

Toto nastavení odpovídá konverzi souřadnic 1:1. Pokud potomci třídy obsluhují zařízení vyžadující kalibraci, je potřeba tuto metodu předefinovat.

7.3.5.31. Metoda TMouseDriver.SetSettings

Metoda **SetSettings** nastavuje veškeré parametry myši z připravené struktury (viz. kapitola 7.2.13).

```
function  SetSettings( const ASettings: TMouseSettings ):
            Boolean; virtual;
```

Parametry:

ASettings	Parametr ASettings obsahuje nastavení parametrů ovladače myši. Všechny položky struktury musí být vyplněny.

Návratové hodnoty:

Pokud metoda **SetSettings** proběhla úspěšně, vrací hodnotu True. V opačném případě vrací hodnotu False..

Poznámky:

Metoda **SetSetttings** vyžaduje, aby byly vyplněny všechny položky struktury ASettings (viz. kapitola 7.2.13), včetně kontrolního součtu. Pokud kontrolní součet nesouhlasí metoda vrátí hodnotu False a v tomto případě není změněn žádný parametr ovladače myši. Metoda interně volá metody **Set_**.

7.3.5.32. Metoda TMouseDriver.GetSetttings

Metoda **GetSettings** uloží do připravené struktury (viz. kapitola 7.2.13) veškeré nastavení ovladače myši.

```
procedure GetSettings( var ASettings: TMouseSettings ); virtual;
```

Parametry:

ASettings	Po provedení metody bude parametr ASettings vyplněn aktuálním nastavení ovladače myši.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Po provedení metody bude parametr ASettings vyplněn aktuálním nastavení ovladače myši. Bude proveden výpočet kontrolního součtu strukturu pomocí funkce **GetChecksum** a tento kontrolní součet bude uložen do položky CheckSum struktury spolu s nastavením ovladače. Metoda interně volá metody **Get_**.

7.3.6. Třída TTimerDriver

Pomocí třída **TTimerDriver** je implementován ovladač časovače. Pomocí ovladače časovače lze naplánovat jednorázovou nebo periodickou událost, která při bude vypršení vyjmuta ze vstupní fronty podobně jako události klávesnice, myši apod.

```
PTimerDriver = ^TTimerDriver;
TTimerDriver = object( TObject )
public
  constructor Init( AMaxTimers: Integer );
  destructor  Done; virtual;
  procedure GetEvent( var Event: TEvent ); virtual;
  function  Schedule( AControl: Pointer; AID: Integer;
              ATime: Longint; AOptions: Word ): Boolean;
  procedure Cancel( AControl: Pointer; AId: Integer );
end;
```

7.3.6.1. Konstruktor TTimerDriver.Init

Konstruktor **Init** provede inicializaci instance třídy.

```
constructor  Init( AMaxEvents: Integer ): Boolean;
```

Parametry:

AMaxTimers	Maximální počet současně naplánovaných časovačů.

Návratové hodnoty:

Konstruktor nevrací žádnou hodnotu.

Poznámky:

Konstruktor inicializuje instanci a na hromadě vytvoří prioritní frontu naplánovaných časovačů o zadané délce.

7.3.6.2. Destruktor TTimerDriver.Done

Destruktor **Done** provede uvolnění prostředků alokovaných třídou.

```
destructor  Done; virtual;
```

Parametry:

Destruktor nemá žádné parametry.

Návratové hodnoty:

Destruktor nevrací žádnou hodnotu.

Poznámky:

Destruktor uvolní z hromady prioritní frontu struktur časovačů alokovaných v konstruktoru.

7.3.6.3. Metoda TTimerDriver.GetEvent

Metoda **GetEvent** předá nejstarší událost a odstraní ji z fronty událostí ovladače časovače.

```
procedure GetEvent( var AEvent: TEvent ); virtual;
```

Parametry:

AEvent	Po provedení metody je do parametru AEvent uložena událost typu evTimer a jsou vyplněny položky TimerId a Control podle parametrů zadaných při naplánování časovače metodou **Schedule**. Pokud v okamžiku volání metody nevypršel žádný nastavený časovače, pak Code hodnotou evNothing.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

7.3.6.4. Metoda TTimerDriver.Schedule

Metoda **Schedule** provádí naplánování časové události.

```
function  Schedule( AControl: Pointer; AID: Integer;
             ATime: Longint; AOptions: Word ): Boolean
```

Parametry:

AControl	Parametr AControl bude po vyjmutí události z fronty pomocí metody GetEvent překopírován do položky Control struktury TEvent. Parametr se odkazuje na komponentu, která událost zpracuje. Parametr může být i nil, a potom je událost časovače zpracována jako broadcast.
AId	Parametr AId jednoznačně identifikuje časovač v rámci komponenty. Parametr je po vyjmutí událost překopírován do

položky TimerId struktury TEvent. Hodnota parametru musí být větší nebo rovna nule.

ATime — Parametr ATime udává zpoždění nebo periodu časovače (dle nastavení parametru AOptions). Časr se zadává v milisekundách a maximální hodnota je 12 hodin.

AOptions — Parametr AOptions určuje chování časovače. Pokud je parametr nastaven na konstantu toOneShot, jedná se o jednorázový časovač, pokud je parametr na staven na konstantu toPeriodic, jedná se o periodický časovač (viz. kapitola 7.1.11).

Návratové hodnoty:

Metoda vrací hodnotu True, pokud se podařilo časovač naplánovat. Pokud metoda vrátí hodnotu False, pak je fronta časovačů plná a časovač nebyl naplánován.

Poznámky:

Jednorázový časovač je automaticky po vypršení odstraněn z fronty. Periodický časovač je po vypršení automaticky znovu naplánován. Předčasné zrušení časovače lze provést pomocí metody Cancel (viz. kapitola 7.3.6.5).

Maximální počet současně naplánovaných časovačů se určuje při vytváření instance TTimerDriver parametrem konstruktoru (viz. kapitola 7.3.6.1).

Pro delší zpoždění než několik desítek minut doporučujeme použít odměřování času pomocí jiný mechanismus než je tento časovač (např. RTC a funkce GetTime, GetDate)

7.3.6.5. Metoda TTimerDriver.Cancel

Metoda **Cancel** provádí předčasné zrušení události časovače.

```
procedure Cancel( AControl: Pointer; AId: Integer );
```

Parametry:

AControl — Parametr AControl musí být nastaven na stejnou hodnotu, jakou měl při naplánování pomocí metody Schedule.

AId — Parametr AId identifikuje rušený časovač, může mít buď hodnotu shodnou při naplánování pomocí metody Schedule. V případě, že obsahuje hodnotu –1, jsou zrušeny všechny časovače, které se stejným parametrem AControl.

Návratové hodnoty:

Metoda nevrací žádné hodnoty.

Poznámky:

Pokud časovač specifikovaný pomocí parametrů AControl a AId nebyl nalezen, pak metoda neprovede žádné zrušení.

7.3.7. Třída TDisplayDriver

Abstraktní třída **TDisplayDriver** definuje obecné rozhraní ovladačů displejů. Potomci této třídy musí předefinovat metody **Initialize**, **Finalize**, **SurfaceChanged** a **Tick**.

```
PDisplayDriver = ^TDisplayDriver;
TDisplayDriver = object( TObject )
public
  Width       : Integer;        { Pocet bodu na radek            }
  Height      : Integer;        { Pocet sloupcu                  }
  BPP         : Byte;           { Pocet bitu na pixel            }
  Surface     : PDrawSurface;   { Kreslici povrch                }
  Caret       : PCaret;         { Stav textoveho kurzoru         }
  Cursor      : PCursor;        { Stav kurzoru mysi              }

  constructor Init( AWidth, AHeight: Integer; ABPP: Byte );
  destructor  Done; virtual;
  function  Initialize: Boolean; virtual;
  procedure Finalize; virtual;
  procedure BeginDraw;
  procedure EndDraw;
  procedure SurfaceChanged; virtual;
  function  Wakeup: Boolean;
  procedure Tick; virtual;

  function  GetContrast: Integer; virtual;
  procedure SetContrast( AValue: Integer ); virtual;
  function  GetBrightness: Integer; virtual;
  procedure SetBrightness( AValue: Integer ); virtual;
  function  GetAutoOffDelay: Integer;
  procedure SetAutoOffDelay( AValue: Integer );
  function  GetAutoOffBrightness: Integer;
  procedure SetAutoOffBrightness( AValue: Integer );

  procedure GetSettings( var ASettings:
              TDisplaySettings ); virtual;
  function  SetSettings( const ASettings:
              TDisplaySettings ): Boolean; virtual;
end;
```

7.3.7.1. Položka TDisplayDriver.Width

Položka **Width** udává šířku displeje v pixelech. Položku nastavuje konstruktor třídy **Init** (viz. kapitola 7.3.7.7) a je určena pouze pro čtení.

```
Width : Integer;
```

7.3.7.2. Položka TDisplayDriver.Height

Položka **Height** udává výšku displeje v pixelech. Položku nastavuje konstruktor třídy **Init** (viz. kapitola 7.3.7.7) a je určena pouze pro čtení.

```
Height : Integer;
```

7.3.7.3. Položka TDisplayDriver.BPP

Položka **BPP** (Bits per pixel) udává počet bitů na jeden pixel, tedy barevnou hloubku. Položku nastavuje konstruktor třídy **Init** (viz. kapitola 7.3.7.7) a je určena pouze pro čtení.

```
BPP : Byte;           { Pocet bitu na pixel            }
```

7.3.7.4. Položka TDisplayDriver.Surface

Položka **Surface** obsahuje odkaz na kreslící povrch displeje (viz. kapitola 7.3.9). Tuto položku nastavuje metoda **Initialize** a je určena pouze ke čtení.

```
Surface : PDrawSurface;   { Kreslici povrch                }
```

7.3.7.5. Položka TDisplayDriver.Caret

Položka **Caret** obsahuje odkaz na ovladač textového kurzoru (viz. kapitola 7.3.14). Tuto položku nastavuje metoda **Initialize** a je určena pouze ke čtení.

```
Caret : PCaret;         { Stav textoveho kurzoru          }
```

7.3.7.6. Položka TDisplayDriver.Cursor

Položka **Caret** obsahuje odkaz na ovladač ukazatele myši (viz. kapitola 7.3.14). Tuto položku nastavuje metoda **Initialize** a je určena pouze ke čtení. Položka může být nastavena na hodnotu **nil**, pokud displej ukazatel myši nezobrazuje.

```
Cursor : PCursor;
```

7.3.7.7. Konstruktor TDisplayDriver.Init

Konstruktor **Init** inicializuje instanci ovladače displeje.

```
constructor Init( AWidth, AHeight: Integer; ABPP: Byte );
```

Parametry:

AWidth	Šířka zobrazovací plochy displeje v pixelech.
AHeight	Výška zobrazovací plochy displeje v pixelech.
ABPP	Počet bitů na bod displeje (barevná hloubka).

Návratové hodnoty:

Konstruktor nevrací žádnou hodnotu

Poznámky:

Konstruktor **Init** inicializuje položky instance na implicitní hodnoty. Automatické snižování jasu displeje při nečinnosti je vypnuto. Položky třídy Surface, Cursor a Caret konstruktor nastavuje na hodnotu **nil**. O jejich inicializaci se musí postarat konstruktor potomka třídy **TDisplayDriver**.

V některých případech není možné, aby již v konstruktoru byly vytvořeny instance TDrawSurface, TCursor a TCaret. Pak je možné tyto inicializace přesunout do metody **Initialize** (viz. kapitola 7.3.7.9).

7.3.7.8. Destruktor TDisplayDriver.Done

Destruktor **Done** uvolňuje prostředky alokované v konstruktoru instance.

```
destructor Done; virtual;
```

Parametry:

Destruktor nemá žádné parametry.

Návratové hodnoty:

Destruktor nevrací žádnou hodnotu

Poznámky:

Destruktor uvolňuje vložené instance kreslícího povrchu (Surface), kurzoru (Caret) a ukazatele myši (Cursor).

7.3.7.9. Metoda TDisplayDriver.Initialize

Metoda **Initialize** provádí inicializaci hardware displeje.

```
function Initialize: Boolean; virtual;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda vrací hodnotu True, pokud se ovladač podařilo úspěšně inicializovat. V opačném případě vrací hodnotu False.

Poznámky:

Metoda **Initialize** ovladače displeje je volána poté, co byla instance vytvořena voláním konstruktoru **Init**.

Přímo ve třídě **TDisplayDriver** je metoda **Initialize** implementována tak, že volá pouze metodu **Initialize** kreslícího povrchu (vložené třídy **TDrawSurface** – viz. položka Surface v kapitola 7.3.7.4). Potomci třídy mohou tuto metodu předefinovat.

Metoda **Initialize** může např. provést nastavení registrů řadiče displeje, přepnutí video-režimu apod. V případě, že vytvoření instancí TDrawSurface, TCursor nebo TCaret je podmíněno přímým přístupem k hardware, může být jejich vytvoření

přesunuto z konstruktoru do této metody.

7.3.7.10. Metoda TDisplayDriver.Finalize

Metoda **Finalize** provádí uvolnění hardwarových prostředků alokovaných metodou **Initialize**.

```
procedure Finalize; virtual;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Metoda **Finalize** je volána těsně před uvolněním instance ovladače displeje pomocí destruktoru **Done**.

Metoda **Finalize** nesmí uvolnit vložené instance Surface, Caret ani Cursor. Tuto činnost provádí až destruktor **Done**.

Přímo ve třídě **TDisplayDriver** je metoda **Finalize** implementována tak, že volá pouze metodu **Finalize** kreslícího povrchu (vložené třídy **TDrawSurface** – viz. položka Surface v kapitola 7.3.7.4). Potomci třídy mohou tuto metodu předefinovat.

7.3.7.11. Metoda TDisplayDriver.BeginDraw

Metoda **BeginDraw** musí být zavolána před jakýmkoli kreslením na displej. Metoda provádí skrytí textového kurzoru a ukazatele myši.

```
procedure BeginDraw;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Pokud jsou v rámci instance displeje vytvořeny instance ukazatele myši a textového kurzoru, pak metoda **BeginDraw** volá příslušné metody pro jejich skrytí (Caret^.**InternalHide** a Mouse^.**Hide**).

Metodu **BeginDraw** je možné volat vícekrát za sebou. Po ukončení vykreslování musí být zavolána metoda **EndDraw** (viz. kapitola 7.3.7.12). Počet volání metody **EndDraw** musí odpovídat počtu volání metody **BeginDraw**.

7.3.7.12. Metoda TDisplayDriver.EndDraw

Metoda **EndDraw** musí být zavolána po ukončení vykreslování na displej. Metoda provádí obnovení (obvykle zobrazení) textového kurzoru a ukazatele myši.

```
procedure EndDraw;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Pokud jsou v rámci instance displeje vytvořeny instance ukazatele myši a textového kurzoru, pak metoda **EndDraw** volá příslušné metody pro jejich skrytí (Caret^.**InternalShow** a Mouse^.**Show**).

Pokud byl mezi voláním metody **BeginDraw** a **EndDraw** změněn stav kreslícího povrchu, metoda **EndDraw** navíc zavolá metodu **SurfaceChanged** (viz. kapitola 7.3.7.13).

7.3.7.13. Metoda TDisplayDriver.SurfaceChanged

Metoda **SurfaceChanged** je volána vždy, když byla provedena změna kreslící povrchu.

```
procedure SurfaceChanged; virtual;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Metoda **SurfaceChanged** je určena pro displeje s vyrovnávací pamětí (videopaměť není mapována do adresového prostoru procesoru). Metoda **SurfaceChanged** musí zajistit přepis změněné části vyrovnávací paměti do paměti řadiče displeje.

Přímo ve třídě **TDisplayDriver** je metoda **SurfaceChanged** prázdná. Potomci třídy mohou tuto metodu předefinovat.

7.3.7.14. Metoda TDisplayDriver.WakeUp

Metoda **WakeUp** probouzí displej z úsporného režimu, tj. režimu se sníženým jasem.

```
function WakeUp: Boolean;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda **WakeUp** vrací hodnotu True, pokud došlo k probuzení displeje z úsporného režimu. V ostatních případech vrací hodnotu False.

Poznámky:

Metoda **WakeUp** je periodicky volána z hlavní smyčky vizualizačního systému, v okamžiku příchodu nějaké vstupní události od uživatele. Pokud je displej v úsporném režimu (snížený jas) dojde k obnovení normálního režimu displeje. V každém případě je restartován časovač pro přechod do úsporného režimu.

7.3.7.15. Metoda TDisplayDriver.Tick

Metoda **Tick** provádí jeden krok automatu displeje.

```
procedure Tick; virtual;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Přímo ve třídě **TDisplayDriver** metoda **Tick** zajišťuje automatické snížení jasu při nečinnosti. Potomci třídy mohou toto metodu předefinovat. Metodu **Tick** je nutné volat v pravidelných intervalech. Obvykle je metoda **Tick** periodicky volána z hlavní smyčky vizulizačního systému.

7.3.7.16. Metoda TDisplayDriver.GetContrast

Metoda **GetContrast** vrací aktuální hodnotu kontrastu displeje.

```
function  GetContrast: Integer; virtual;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda vrací aktuální hodnotu kontrastu displeje. Vrácená hodnota je poplatná implementaci ovladače konkrétního displeje.

Poznámky:

Potomci třídy **TDisplayDriver** mohou metodu **GetContrast** předefinovat, tak aby vracela správnou hodnotu kontrastu. Metoda **GetContrast** implementovaná třídou **TDisplayDriver** vrací vždy hodnotu nula.

7.3.7.17. Metoda TDisplayDriver.SetContrast

Metoda **SetContrast** slouží pro nastavení aktuální hodnoty kontrastu displeje.

```
procedure SetContrast( AValue: Integer ); virtual;
```

Parametry:

AValue	Požadovaná hodnota kontrastu. Hodnota je poplatná implementaci konkrétního ovladače displeje.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Potomci třídy **TDisplayDriver** mohou metodu **SetContrast** předefinovat, tak aby správně nastavovala kontrast displeje. Metoda **SetContrast** implementovaná třídou **TDisplayDriver** je prázdná a neprovádí žádnou činnost.

7.3.7.18. Metoda TDisplayDriver.GetBrightness

Metoda **GetBrightness** vrací aktuální hodnotu jasu (úroveň podsvícení) displeje.

```
function  GetBrightness: Integer; virtual;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda vrací aktuální hodnotu jasu displeje. Vrácená hodnota je poplatná implementaci ovladače konkrétního displeje.

Poznámky:

Potomci třídy **TDisplayDriver** mohou metodu **GetBrightness** předefinovat, tak aby vracela správnou hodnotu jasu. Metoda **GetBrightness** implementovaná třídou **TDisplayDriver** vrací vždy hodnotu nula.

7.3.7.19. Metoda TDisplayDriver.SetBrightness

Metoda **SetBrightness** slouží pro nastavení aktuální hodnoty jasu (úrovně podsvícení) displeje.

```
procedure SetBrightness( AValue: Integer ); virtual;
```

Parametry:

AValue	Požadovaná hodnota jasu. Hodnota je poplatná implementaci konkrétního ovladače displeje.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Potomci třídy **TDisplayDriver** mohou metodu **SetBrightness** předefinovat, tak aby správně nastavovala jas displeje. Metoda **SetBrightness** implementovaná třídou **TDisplayDriver** je prázdná a neprovádí žádnou činnost.

7.3.7.20. Metoda TDisplayDriver.GetAutoOffDelay

Metoda **GetAutoOffDelay** vrací aktuálně nastavený čas, po kterém dojde při nečinnosti k automatickému snížení jasu (úrovně podsvícení) displeje na úroveň definovanou pomocí metody **SetAutoOffBrightness** (viz. kapitola 7.3.7.23)

```
function  GetAutoOffDelay: Integer;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda vrací počet sekund, po kterém dojde při nečinnosti uživatele k automatickému snížení jasu. Pokud je tato funkce vypnuta, metoda vrací hodnotu nula.

Poznámky:

7.3.7.21. Metoda TDisplayDriver.SetAutoOffDelay

Metoda **SetAutoOffDelay** slouží k nastavení času, po kterém dojde při nečinnosti k automatickému snížení jasu (úrovně podsvícení) displeje na úroveň definovanou pomocí metody **SetAutoOffBrightness** (viz. kapitola 7.3.7.23)

```
procedure SetAutoOffDelay( AValue: Integer );
```

Parametry:

AValue	Počet sekund, po kterých při nečinnosti uživatele dojde ke snížení jasu displeje. V případě, že parametr obsahuje hodnotu nula, pak je funkce automatického snižovaní jasu vypnuta.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

7.3.7.22. Metoda TDisplayDriver.GetAutoOffBrightness

Metoda **GetAutoOffBrightness** vrací úroveň jasu (podsvícení) displeje, která bude automaticky nastavena po době určené metodou **SetAutoOffDelay** (viz. kapitola 7.3.7.21).

```
function  GetAutoOffBrightness: Integer;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda vrací hodnotu jasu displeje. Vrácená hodnota je poplatná implementaci ovladače konkrétního displeje.

Poznámky:

7.3.7.23. Metoda TDisplayDriver.SetAutoOffBrightness

Metoda **SetAutoOffBrightness** slouží k nastavení úrovně jasu (podsvícení) displeje, která bude automaticky nastavena po době určené metodou **SetAutoOffDelay** (viz. kapitola 7.3.7.21).

```
function  GetAutoOffBrightness: Integer;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda vrací hodnotu jasu displeje. Vrácená hodnota je poplatná implementaci ovladače konkrétního displeje.

Poznámky:

7.3.7.24. Metoda TDisplayDriver.GetSettings

Metoda **GetSettings** uloží do připravené struktury (viz. kapitola 7.2.14) veškeré nastavení ovladače displeje.

```
procedure GetSettings( var ASettings: TDisplaySettings ); virtual;
```

Parametry:

ASettings	Po provedení metody bude parametr ASettings vyplněn aktuálním nastavení ovladače displeje.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Po provedení metody bude parametr ASettings vyplněn aktuálním nastavení ovladače displeje. Bude proveden výpočet kontrolního součtu strukturu pomocí funkce **GetChecksum** a tento kontrolní součet bude uložen do položky CheckSum struktury spolu s nastavením ovladače. Metoda interně volá metody **Get_**.

7.3.7.25. Metoda TDisplayDriver.SetSettings

Metoda **SetSettings** nastavuje veškeré parametry ovladače displeje z připravené struktury (viz. kapitola 7.2.13).

```
function  SetSettings( const ASettings: TDisplaySettings ):
            Boolean; virtual;
```

Parametry:

ASettings	Parametr ASettings obsahuje nastavení parametrů ovladače displeje. Všechny položky struktury musí být vyplněny.

Návratové hodnoty:

Pokud metoda **SetSettings** proběhla úspěšně, vrací hodnotu True. V opačném případě vrací hodnotu False..

Poznámky:

Metoda **SetSetttings** vyžaduje, aby byly vyplněny všechny položky struktury ASettings (viz. kapitola 7.2.13), včetně kontrolního součtu. Pokud kontrolní součet nesouhlasí metoda vrátí hodnotu False a v tomto případě není změněn žádný parametr ovladače displeje. Metoda interně volá metody **Set_**.

7.3.8. Třída TCachedDisplayDriver

Abstraktní třída **TCachedDisplayDriver** je bázovou třídou pro všechny ovladače displejů, jejichž videopaměť není mapována přímo do adresového prostoru procesoru. Třída řeší problém přepisování virtuální videopaměti (kopie videopaměti v paměti RAM) do fyzické videopaměti v řadiči displeje.

```
PCachedDisplayDriver = ^TCachedDisplayDriver;
TCachedDisplayDriver = object( TDisplayDriver )
public
  Timeout1  : Integer;
  Timeout2  : Integer;

  constructor Init( AWidth, AHeight: Integer; ABPP: Byte );
  destructor  Done; virtual;
  procedure SurfaceChanged; virtual;
  procedure Tick; virtual;

  procedure SetTimeouts( ATimeout1, ATimeout2: Integer );
  procedure Refresh( const R: TRect ); virtual; {abstract;}
end;
```

Třída poskytuje abstraktní metodu **Refresh**, kterou musí potomci předefinovat, tak aby přepisovala data z kopie videopaměti do fyzické videopaměti v řadiči displeje.

Metoda **Refresh** je volána tehdy, když dojde k zápisu do kopie videopaměti. Volání této metody je řízeno pomocí dvou časovačů (T1 a T2) a metod **SurfaceChanged** a **Tick** (viz. kapitola 7.3.8.1)

7.3.8.1. Časování kopírování videopaměti

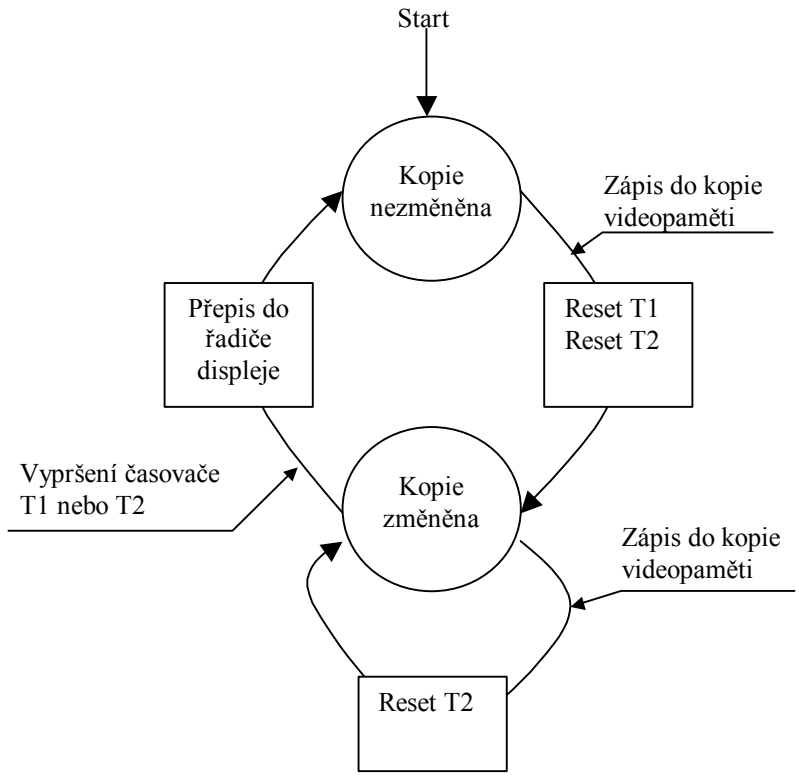
Pokud bychom okamžitě po provedení změny ve videopaměti změněnou část kopírovali do paměti řadiče displeje, pak by mohlo docházet ke zbytečnému plýtvání s procesorovým časem. Třída **TCachedDisplayDriver** provádí kopírování po změně videopaměti s mírnou časovou prodlevou, která zajistí přepis ve vhodnějších okamžicích.

Kopírování je řízeno pomocí časovaču T1 a T2. Časovač T2 zajišťuje co nejrychlejší překopírování změn provedených ve vyrovnávací paměti, např. po stisku klávesy apod. Časovač T2 zamezuje zbytečnému kopírování. Je resetován při každé změně v kopii videopaměti, a až okamžiku jeho vypršení, tj. po nastavené době po poslední provedené změně spuštěno kopírování.

Časovač T1 zajišťuje překopírování změn v případě, že se obsah vyrovnávací paměti neustále mění (tj. časovač T2 je neustále resetován a nemůže tím pádem vypršet).

Algoritmus přepisování kopie videopaměti do paměti řadiče popisuje stavový diagram

níže:



7.3.8.2. Položka TCachedDisplayDriver.Timeout1

Položka **Timeout1** obsahuje aktuální nastavení časovače T1 (viz. kapitola 7.3.8.1), který řídí přepisování kopie videopaměti do paměti řadiče displeje. Hodnota je v milisekundách. Položka je určena pouze pro čtení a pro její nastavení slouží metoda **SetTimeouts** (viz. kapitola 7.3.8.4)

7.3.8.3. Položka TCachedDisplayDriver.Timeout2

Položka **Timeout2** obsahuje aktuální nastavení časovače T2 (viz. kapitola 7.3.8.1), který řídí přepisování kopie videopaměti do paměti řadiče displeje. Hodnota je v milisekundách. Položka je určena pouze pro čtení a pro její nastavení slouží metoda **SetTimeouts** (viz. kapitola 7.3.8.4)

7.3.8.4. Metoda TCachedDisplayDriver.SetTimeouts

Metoda **SetTimeouts** slouží k nastavení časovaču T1 a T2 (viz. kapitola 7.3.8.1), které řídí přepisování kopie videopaměti do paměti řadiče displeje.

```
procedure SetTimeouts( ATimeout1, ATimeout2: Integer );
```

Parametry:

ATimeout1	Doba vypršení časovače T1. Hodnota se zadává v milisekundách.

ATimeout2	Doba vypršení časovače T2. Hodnota se zadává v milisekundách.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

7.3.8.5. Metoda TCachedDisplayDriver.Refresh

Abstraktní metoda **Refresh** slouží k překopírování části kopie videopaměti do paměti řadiče displeje.

```
procedure Refresh( const R: TRect ); virtual;
```

Parametry:

R	Obdélník ohraničující oblast videopaměti, kterou je potřeba přenést do paměti řadiče displeje.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Přímo ve třídě **TCachdeDisplayDriver** je metoda **Refresh** prázdná. Potomci třídy musí tuto metodu předefinovat.

7.3.9. Třída TDrawSurface

Abstraktní třída TDrawSurface definuje rozhraní obecného kreslícího povrch (tj. metody pro kreslení do videopaměti).

```
PDrawSurface = ^TDrawSurface;
TDrawSurface = object( TObject )
public
  Changes    : TRect;

  constructor Init;
  destructor Done; virtual;

  function ClipLine( const AParams: TDrawParams;
            var AResult: TLineBres ): Boolean;


  function  Initialize: Boolean; virtual;
  procedure Finalize; virtual;

  function  Clear( const AParams: TDrawParams ): TDrawStatus;
             virtual;
```

```
    function  Copy( const AParams: TDrawParams ): TDrawStatus;
                virtual;
    function  Scroll( const AParams: TDrawParams ): TDrawStatus;
                virtual;
    function  DrawText( const AParams: TDrawParams ): TDrawStatus;
                virtual;

    function  DrawPoint( const AParams: TDrawParams ): TDrawStatus;
                virtual;
    function  FillRect( const AParams: TDrawParams ): TDrawStatus;
                virtual;
    function  DrawBitmap( const AParams: TDrawParams ): TDrawStatus;
                virtual;
    function  CopyToBitmap( const AParams: TDrawParams ):
                 TDrawStatus; virtual;

    function  DrawLine( const AParams: TDrawParams ): TDrawStatus;
                 virtual;
    function  DrawCircle( const AParams: TDrawParams ): TDrawStatus;
                 virtual;
    function  DrawEllipse( const AParams: TDrawParams ): TDrawStatus;
                 virtual;
    function  DrawArc( const AParams: TDrawParams ): TDrawStatus;
                 virtual;
    function  FillCircle( const AParams: TDrawParams ): TDrawStatus;
                 virtual;
    function  FillEllipse( const AParams: TDrawParams ): TDrawStatus;
                 virtual;
    function  FillArc( const AParams: TDrawParams ): TDrawStatus;
                 virtual;
  end;
```

Potomci třídy **TDrawSurface** musí povinně předefinovat metody **Clear**, **Copy**, **Scroll** a **DrawText**. Pokud se jedná o grafický povrch, musí navíc předefinovat metody **DrawPoint**, **FillRect**, **DrawBitmap** a **CopyToBitmap**.

Metody **DrawLine**, **DrawCircle**, **DrawEllipse**, **DrawArc**, **FillCircle**, **FillEllipse** a **FillArc** není nutné předefinovávat. Přímo ve třídě **TDrawSurface** jsou implementovány tak, že interně volají metodu **DrawPoint** příp. **FillRect**. Nicméně takováto implementace je v mnohých případech pomalá. Proto doporučujeme optimalizovat alespoň metodu **DrawLine** pro rychlé vykreslení horizontálních a vertikálních úseček.

7.3.9.1. Položka TDrawSurface.Changes

Položka Changes ohraničuje změny provedené ve videopaměti. Jedná se o obdélník který je při zápisu do videopaměti (volání jedné z vykreslovacích metod) vždy rozšířen tak, aby pokrýval celou oblast ve které byly provedeny změny.

```
Changes : TRect;
```

7.3.9.2. Konstruktor TDrawSurface.Init

Konstruktor **Init** provádí inicializaci instance třídy.

```
constructor Init;
```

Parametry:

Konstruktor nemá žádné parametry.

Návratové hodnoty:

Konstruktor nevrací žádnou hodnotu.

Poznámky:

Přímo ve třídě **TDrawSurface** je konstruktor prázdný.

7.3.9.3. Destruktor TDrawSurface.Done

Destruktor **Done** provádí uvolnění prostředků alokovaných v konstruktoru instance.

```
destructor Done;
```

Parametry:

Destruktor nemá žádné parametry.

Návratové hodnoty:

Destruktor nevrací žádnou hodnotu.

Poznámky:

Přímo ve třídě **TDrawSurface** je destruktor prázdný.

7.3.9.4. Metoda TDrawSurface.Initialize

Metoda **Initialize** provádí inicializaci kreslícího povrchu.

```
function Initialize: Boolean; virtual;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda vrací hodnotu True, pokud se kreslící povrch podařilo úspěšně inicializovat. V opačném případě vrací hodnotu False.

Poznámky:

Metoda **Initialize** kreslícího povrchu je volána poté, co byla instance vytvořena voláním konstruktoru **Init**. Konkrétně je volána z metody **Initialize** třídy

TDisplayDriver.

Přímo ve třídě **TDrawSurface** je metoda **Initialize** prázdná. Potomci třídy mohou tuto metodu v případě potřeby předefinovat.

7.3.9.5. Metoda TDrawSurface.Finalize

Metoda **Finalize** provádí uvolnění prostředků alokovaných metodou **Initialize**.

```
procedure Finalize; virtual;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Metoda **Finalize** je volána těsně před uvolněním instance kreslícího povrchu pomocí destruktoru **Done**.

Přímo ve třídě **TDrawSurface** je metoda **Finalize** prázdná. Potomci třídy mohou tuto metodu v případě potřeby předefinovat.

7.3.9.6. Metoda TDrawSurface.ClipLine

Metoda **ClipLine** slouží k ořezání úsečky obdelníkem. Vstupem této funkce jsou koncové body úsečky, ořezávací obdélník a výstupem jsou parametry Bresenhamova algoritmu pro vykreslení úsečky.

```
function  ClipLine( const AParams: TDrawParams;
            var AResult: TLineBres ): Boolean;
```

Parametry:

AParams	Parametry jsou shodné s parametry metody DrawLine.

Návratové hodnoty:

Metoda vrací hodnotu False, pokud je úsečka úplně ořezána a není potřeba vykreslit žádnou její část. V případě, že metoda vrátí hodnotu True, je úsečka alespoň z částí viditelná a do parametru AResult jsou uloženy parametry Bresenhamova algoritmu pro vykreslování úseček.

Poznámky:

Metoda slouží pro vykreslování úseček a je využívána metodami **DrawLine** potomků třídy **TDrawSurface**. Více. informací naleznete ve zdrojovém textu knihovny.

7.3.9.7. Metoda TDrawSurface.Clear

Metoda **Clear** nastaví všechny body kreslícího povrchu na hodnotu nula (tj. černou barvu)

```
function  Clear( const AParams: TDrawParams ): TDrawStatus;
            virtual;
```

Parametry:

AParams	Parametr je metodou ignorován.

Návratové hodnoty:

V případě úspěchu vrací metoda hodnotu dsSuccess. V případě, že funkce není implementovaná vrací metoda hodnotu dsNotImpl. V případě neplatných parametrů vrací metoda hodnotu dsInvPar.

Poznámky:

Položka **Text** parametru AParams ukazuje na vykreslovaný text. Délku textu lze určit pomocí položky **Length** parametru AParams. Pokud metoda v textu narazí na nulový znak, vykreslování textu ukončí.

Přímo ve třídě **TDrawSurface** je metoda **DrawPoint** implementována tak, že vrací hodnotu dsNotImpl. Potomci třídy musí tuto metodu předefinovat.

7.3.9.8. Metoda TDrawSurface.Copy

Tato metoda není v současné době implementovaná.

Metoda **Copy** kopíruje obsah obdélníkové oblast kreslícího povrchu na jiné místo.

```
function  Copy( const AParams: TDrawParams ): TDrawStatus;
            virtual;
```

Parametry:

AParams	Parametr AParams určuje jak a co se má kopírovat. Je potřeba vyplnit následující položky struktury: Points[0] Points[1] Points[2]

Návratové hodnoty:

V případě úspěchu vrací metoda hodnotu dsSuccess. V případě, že funkce není implementovaná vrací metoda hodnotu dsNotImpl. V případě neplatných parametrů

vrací metoda hodnotu dsInvPar.

Poznámky:

Přímo ve třídě **TDrawSurface** je metoda **Copy** implementována tak, že vrací hodnotu dsNotImpl. Potomci třídy musí tuto metodu předefinovat.

7.3.9.9. Metoda TDrawSurface.Scroll

Tato metoda není v současné době implementovaná.

Metoda **Scroll** roluje obsah obdélníkové oblast kreslícího povrchu o zadaný počet pixelů.

```
function  Copy( const AParams: TDrawParams ): TDrawStatus;
            virtual;
```

Parametry:

AParams	Parametr AParams určuje jak a co se má kopírovat. Je potřeba vyplnit následující položky struktury: Clip Ořezávací oblast, která zároveň definuje rolovanou oblast Points[0] .X Pusunutí v horizontálním směru Points[0].Y Posunutí ve vertikálním směru

Návratové hodnoty:

V případě úspěchu vrací metoda hodnotu dsSuccess. V případě, že funkce není implementovaná vrací metoda hodnotu dsNotImpl. V případě neplatných parametrů vrací metoda hodnotu dsInvPar.

Poznámky:

Přímo ve třídě **TDrawSurface** je metoda **Scroll** implementována tak, že vrací hodnotu dsNotImpl. Potomci třídy musí tuto metodu předefinovat.

7.3.9.10. Metoda TDrawSurface.DrawText

Metoda **DrawText** vykresluje zadaný text daným fontem.

```
function  DrawText( const AParams: TDrawParams ): TDrawStatus;
            virtual;
```

Parametry:

AParams	Parametr AParams určuje jak a kde se má text vykreslit. Je potřeba vyplnit následující položky struktury:

Points[0]	Souřadnice bodu
Clip	Souřadnice ořezávacího obdélníku
Rop	Operace s pozadím
Font	Definice pera, kterým bude bod vykreslen
Text	Ukazatel na vykreslovaný text
Length	Délka textu

Návratové hodnoty:

V případě úspěchu vrací metoda hodnotu dsSuccess. V případě, že funkce není implementovaná vrací metoda hodnotu dsNotImpl. V případě neplatných parametrů vrací metoda hodnotu dsInvPar.

Poznámky:

Položka **Text** parametru AParams ukazuje na vykreslovaný text. Délku textu lze určit pomocí položky **Length** parametru AParams. Pokud metoda v textu narazí na nulový znak, vykreslování textu ukončí.

Přímo ve třídě **TDrawSurface** je metoda **Clear** implementována tak, že vrací hodnotu dsNotImpl. Potomci třídy musí tuto metodu předefinovat.

7.3.9.11. Metoda TDrawSurface.DrawPoint

Metoda **DrawPoint** vykresluje jeden bod daného stylu a barvy na zadaných souřadnicích.

```
function  DrawPoint( const AParams: TDrawParams ): TDrawStatus;
             virtual;
```

Parametry:

AParams — Parametr AParams určuje jak a kde se má bod vykreslit. Je potřeba vyplnit následující položky struktury:

Points[0]	Souřadnice bodu
Clip	Souřadnice ořezávacího obdélníku
Rop	Operace s pozadím
Pen	Definice pera, kterým bude bod vykreslen

Návratové hodnoty:

V případě úspěchu vrací metoda hodnotu dsSuccess. V případě, že funkce není implementovaná vrací metoda hodnotu dsNotImpl. V případě neplatných parametrů vrací metoda hodnotu dsInvPar.

Poznámky:

Přímo ve třídě **TDrawSurface** je metoda **DrawText** implementována tak, že vrací hodnotu dsNotImpl. Potomci třídy musí tuto metodu předefinovat.

7.3.9.12. Metoda TDrawSurface.FillRect

Metoda **FillRect** vyplní obdélníkovou oblast zadaným štetcem.

```
function  FillRect( const AParams: TDrawParams ): TDrawStatus;
            virtual;
```

Parametry:

AParams	Parametr AParams určuje jak a kde se má obdélník vyplnit. Je potřeba vyplnit následující položky struktury:	
	Points[0]	Levý horní roh obdélníku
	Points[1]	Pravý dolní roh obdélníku (včetně)
	Clip	Souřadnice ořezávacího obdélníku
	Rop	Operace s pozadím
	Brush	Definice štětce, kterým bude obdélník vyplněn

Návratové hodnoty:

V případě úspěchu vrací metoda hodnotu dsSuccess. V případě, že funkce není implementovaná vrací metoda hodnotu dsNotImpl. V případě neplatných parametrů vrací metoda hodnotu dsInvPar.

Poznámky:

Přímo ve třídě **TDrawSurface** je metoda **FillRect** implementována tak, že vrací hodnotu dsNotImpl. Potomci třídy musí tuto metodu předefinovat.

7.3.9.13. Metoda TDrawSurface.DrawBitmap

Metoda **DrawBitmap** slouží k vykreslení bitmapy na zadané souřadnice kreslícího povrchu.

```
function  DrawBitmap( const AParams: TDrawParams ): TDrawStatus;
            virtual;
```

Parametry:

AParams	Parametr AParams určuje jak a kde se má bitmapa vykreslit. Je potřeba vyplnit následující položky struktury:	
	Points[0]	Levý horní roh bitmapy
	Clip	Souřadnice ořezávacího obdélníku oblast
	ROP	Operace s pozadím

BitmapUkazatel na strukturu bitmapy

Návratové hodnoty:

V případě úspěchu vrací metoda hodnotu dsSuccess. V případě, že funkce není implementovaná vrací metoda hodnotu dsNotImpl. V případě neplatných parametrů vrací metoda hodnotu dsInvPar.

Poznámky:

Přímo ve třídě **TDrawSurface** je metoda **DrawBitmap** implementována tak, že vrací hodnotu dsNotImpl. Potomci třídy musí tuto metodu předefinovat.

7.3.9.14. Metoda TDrawSurface.CopyToBitmap

Metoda **CopyToBitmap** slouží k překopírování části kreslicího povrchu do připravené bitmapy.

```
function  CopyToBitmap( const AParams: TDrawParams ): TDrawStatus;
            virtual;
```

Parametry:

AParams Parametr AParams určuje chování metody.

Points[0]	Levý horní roh bitmapy kopírované oblasti
Clip	Souřadnice ořezávacího oblasti
BitmapUkazatel na strukturu bitmapy	

Návratové hodnoty:

V případě úspěchu vrací metoda hodnotu dsSuccess. V případě, že funkce není implementovaná vrací metoda hodnotu dsNotImpl. V případě neplatných parametrů vrací metoda hodnotu dsInvPar.

Poznámky:

Do bitmapy se kopíruje oblast daná bodem Point[0] a rozměry bitmapy. Pokud je nějaká část této oblasti mimo ořezávací oblast, pak korespondující část bitmapy zůstane nedotčena.

Přímo ve třídě **TDrawSurface** je metoda **CopyToBitmap** implementována tak, že vrací hodnotu dsNotImpl. Potomci třídy musí tuto metodu předefinovat.

7.3.9.15. Metoda TDrawSurface.DrawLine

Metoda **DrawLine** slouží k vykreslení úsečky na zadané souřadnice kreslícího povrchu.

```
function  DrawLine( const AParams: TDrawParams ): TDrawStatus;
            virtual;
```

Parametry:

AParams	Parametr AParams určuje jak a kde se má úsečka vykreslit. Je potřeba vyplnit následující položky struktury:

Points[0]	Počáteční bod úsečky
Points[1]	Koncový bod úsečky
Clip	Souřadnice ořezávacího obdélníku
ROP	Operace s pozadím
Pen	Styl pera

Návratové hodnoty:

V případě úspěchu vrací metoda hodnotu dsSuccess. V případě, že funkce není implementovaná vrací metoda hodnotu dsNotImpl. V případě neplatných parametrů vrací metoda hodnotu dsInvPar.

Poznámky:

Přímo ve třídě **TDrawSurface** je metoda **DrawLine** implementována tak, že volá opakovaně metodu **DrawPoint**. Potomci třídy by měli tuto metodu předefinovat, tak aby vykreslovala úsečku optimálně vzhledem k vlastnostem kreslícího povrchu..

7.3.9.16. Metoda TDrawSurface.DrawCircle

Metoda **DrawCircle** slouží k vykreslení kružnice na zadané souřadnice kreslícího povrchu.

```
function  DrawCircle( const AParams: TDrawParams ): TDrawStatus;
            virtual;
```

Parametry:

AParams	Parametr AParams určuje jak a kde se má kružnice vykreslit. Je potřeba vyplnit následující položky struktury:

Points[0]	Střed kružnice
Points[1].X	Poloměr kružnice
Clip	Souřadnice ořezávacího obdélníku
ROP	Operace s pozadím
Pen	Styl pera

Návratové hodnoty:

V případě úspěchu vrací metoda hodnotu dsSuccess. V případě, že funkce není

implementovaná vrací metoda hodnotu dsNotImpl. V případě neplatných parametrů vrací metoda hodnotu dsInvPar.

Poznámky:

Přímo ve třídě **TDrawSurface** je metoda **DrawCircle** implementována tak, že volá opakovaně metodu **DrawPoint**. Potomci třídy mohou tuto metodu předefinovat, tak aby vykreslovala kružnici optimálně vzhledem k vlastnostem kreslícího povrchu..

7.3.9.17. Metoda TDrawSurface.DrawEllipse

Metoda **DrawEllipse** slouží k vykreslení elipsy na zadané souřadnice kreslícího povrchu.

```
function  DrawEllipse( const AParams: TDrawParams ): TDrawStatus;
            virtual;
```

Parametry:

AParams	Parametr AParams určuje jak a kde se má elipsa vykreslit. Je potřeba vyplnit následující položky struktury:	
	Points[0]	Levý horní roh obdélníku obklopujícího elipsu
	Points[1]	Pravý dolní roh obdélníku obklopujícího elipsu
	Clip	Souřadnice ořezávacího obdélníku
	ROP	Operace s pozadím
	Pen	Styl pera

Návratové hodnoty:

V případě úspěchu vrací metoda hodnotu dsSuccess. V případě, že funkce není implementovaná vrací metoda hodnotu dsNotImpl. V případě neplatných parametrů vrací metoda hodnotu dsInvPar.

Poznámky:

Přímo ve třídě **TDrawSurface** je metoda **DrawEllipse** implementována tak, že volá opakovaně metodu **DrawPoint**. Potomci třídy mohou tuto metodu předefinovat, tak aby vykreslovala elipsu optimálně vzhledem k vlastnostem kreslícího povrchu..

7.3.9.18. Metoda TDrawSurface.DrawArc

Tato metoda není v současné době implementovaná.

Metoda **DrawArc** slouží k vykreslení části elipsy (oblouku) na zadané souřadnice kreslícího povrchu.

```
function  DrawArc( const AParams: TDrawParams ): TDrawStatus;
            virtual;
```

7.3.9.19. Metoda TDrawSurface.FillCircle

Metoda **FillCircle** slouží k vyplnění kružnice na zadaných souřadnicích kreslícího povrchu.

```
function  FillCircle( const AParams: TDrawParams ): TDrawStatus;
            virtual;
```

Parametry:

AParams	Parametr AParams určuje jak a kde se má kružnice vykreslit. Je potřeba vyplnit následující položky struktury:

Points[0]	Střed kružnice
Points[1].X	Poloměr kružnice
Clip	Souřadnice ořezávacího obdélníku
ROP	Operace s pozadím
Brush	Styl štětce

Návratové hodnoty:

V případě úspěchu vrací metoda hodnotu dsSuccess. V případě, že funkce není implementovaná vrací metoda hodnotu dsNotImpl. V případě neplatných parametrů vrací metoda hodnotu dsInvPar.

Poznámky:

Přímo ve třídě **TDrawSurface** je metoda **FillCircle** implementována tak, že volá opakovaně metodu **FillRect**. Potomci třídy mohou tuto metodu předefinovat, tak aby vyplňovala kružnici optimálně vzhledem k vlastnostem kreslícího povrchu..

7.3.9.20. Metoda TDrawSurface.FillEllipse

Metoda **FillEllipse** slouží k vyplnění elipsy na zadaných souřadnicích kreslícího povrchu.

```
function  FillEllipse( const AParams: TDrawParams ): TDrawStatus;
            virtual;
```

Parametry:

AParams	Parametr AParams určuje jak a kde se má elipsa vykreslit. Je potřeba vyplnit následující položky struktury:

Points[0]	Levý horní roh obdélníku obklopujícího elipsu
Points[1]	Pravý dolní roh obdélníku obklopujícího elipsu
Clip	Souřadnice ořezávacího obdélníku
ROP	Operace s pozadím
Brush	Styl štětce

Návratové hodnoty:

V případě úspěchu vrací metoda hodnotu dsSuccess. V případě, že funkce není implementovaná vrací metoda hodnotu dsNotImpl. V případě neplatných parametrů vrací metoda hodnotu dsInvPar.

Poznámky:

Přímo ve třídě **TDrawSurface** je metoda **FillEllipse** implementována tak, že volá opakovaně metodu **FillRect**. Potomci třídy mohou tuto metodu předefinovat, tak aby vyplňovala elipsu optimálně vzhledem k vlastnostem kreslícího povrchu..

7.3.9.21. Metoda TDrawSurface.FillArc

Tato metoda není v současné době implementovaná.

Metoda **FillArc** slouží k vyplnění elipsy na zadaných souřadnicích kreslícího povrchu.

```
function  FillArc( const AParams: TDrawParams ): TDrawStatus;
            virtual;
```

7.3.10. Třída T1BPPDrawSurface

Třída **T1BPPDrawSurface** implementuje kreslící povrch s jedním bitem na pixel

```
P1BPPDrawSurface = ^T1BPPDrawSurface;
T1BPPDrawSurface = object( TDrawSurface )
public
  { ASize.X musi byt nasobkem 8 }
  constructor Init( AWindowPtr: Pointer; ARowSize: Word );

  ...
end;
```

7.3.10.1. Konstruktor T1BPPDrawSurface.Init

Konstruktor **Init** inicializuje instanci kreslícího povrchu.

```
constructor Init( AWindowPtr: Pointer; ARowSize: Word );
```

Parametry:

AWindowPtr	Ukazatel na začátek videopaměti.
ARowSize	Šířka jednoho řádku videopaměti v bajtech. Tj. obvykle počet pixelů na řádku dělený osmi.

Návratové hodnoty:

Konstruktor nevrací žádnou hodnotu.

Poznámky:

7.3.11. Třída T8BPPDrawSurface

Třída **T8BPPDrawSurface** implementuje kreslící povrch s osmi bity na pixel.

```
P8BPPDrawSurface = ^T8BPPDrawSurface;
T8BPPDrawSurface = object( TDrawSurface )
public
  ...

  constructor Init( AWindowPtr: Pointer; ARowSize: Word;
    AWinSize: Word; AWinGranul: Byte;
    AWinFuncPtr: Pointer; AWinFuncCtx: Pointer );

  ...
end;
```

7.3.11.1. Konstruktor T8BPPDrawSurface.Init

Konstruktor **Init** provádí inicializaci instance třídy.

```
constructor Init( AWindowPtr: Pointer; ARowSize: Word;
  AWinSize: Word; AWinGranul: Byte;
  AWinFuncPtr: Pointer; AWinFuncCtx: Pointer );
```

Parametry:

AWindowPtr	Ukazatel na začátek okna videopaměti.
ARowSize	Šířka jednoho řádku videopaměti v bajtech. Tj. počet pixelů na řádku.
AWinSize	Velikost okna videopaměti v KB.
AWinGranul	Granularita okna v KB.
AWinFuncPtr	Ukazatel na funkci přepínající okna
AWinFuncCtx	Pomocná hodnota předávaná funkci přepínající okna.

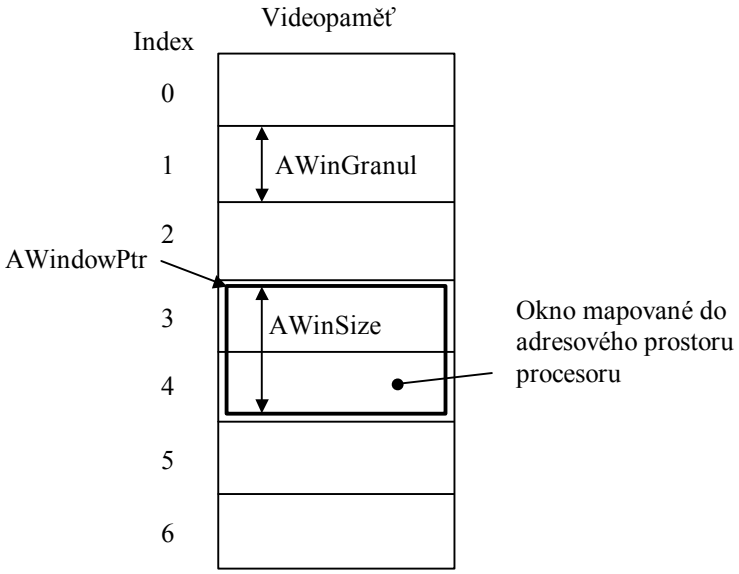
Návratové hodnoty:

Konstruktor nevrací žádnou hodnotu.

Poznámky:

Třída **T8BPPDrawSurface** umožňuje zapisovat do videopaměti, která není do adresového prostoru namapovaná celá, ale pouze po částech. V adresovém prostoru je namapovaná pouze část videopaměti – tzv. okno. Obvykle pomocí speciálních registrů řadiče displeje se určí, která část videopaměti má být v tomto okně vidět. Okno má určitou velikost (parametr AWinSize) a nachází v adresovém prostoru procesoru na definovaném místě (parametr AWindowPtr). Řadič displeje umožňuje přepínat okno po určitých krocích (parametr AWinGranul), např. okno velké 32kB, lze přepínat s krokem 16kB.

Funkci parametrů konstruktoru AWindowPtr, AWinSize, AWinGranul ukazuje následující obrázek:



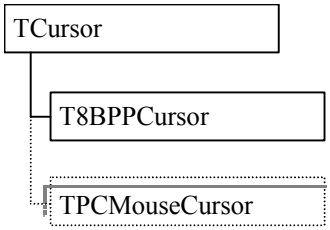
Pro přepínání okna je potřeba definovat speciální funkci (parametr AWinFuncPtr), která je volána ve vhodných okamžicích z vykreslovacích metod a musí zajistit správné přepnutí okna. Před voláním této funkce je na zásobník zkopírován parametr AWinFuncCtx a do registru AX je uložen index okna. **Pozor: Register DS je při vstupu do funkce nedefinovaný, to znamená, že ve funkci nelze přistupovat ke globálním proměnným.** Po provedení přepnutí okna musí funkce odstranit ze zásobníku parametr AWinFuncCtx. Funkce po návratu nesmí změnit žádné registry.

Viz. následující příklad:

```
{$S-}
procedure WindowSwitch( AWinFuncCtx: Pointer ); far; assembler;
asm
        PUSH    DX
        MOV     DX, regWindow { Register indexu okna }
        OUT     DX, AX
        POP     DX
end;
{$S+}
```

7.3.12. Třída TCursor

Abstraktní třída **TCursor** definuje obecné rozhraní ukazatele myši. Ukazatel myši je malá šipka zobrazovaná v aktuálních souřadnicích myši. Instance třídy ukazatele myši se vytváří jen v případě, že příslušný terminál je vybaven polohovacím zařízení jako je myš, trackball apod. (např. simulátor terminálu na PC). Běžné terminály fy SofCon ukazatel myši nezobrazují.



```
PCursor = ^TCursor;
TCursor = object( TObject )
public
  procedure SetPosition( AX, AY: Integer ); virtual;
  procedure Show; virtual;
  procedure Hide; virtual;
end;
```

Pozice ukazatel myši je automaticky nastavována ovladačem myši **TMouseDriver**, viz. kapitola 7.3.5.

7.3.12.1. Metoda TCursor.SetPosition

Metoda **SetPosition** nastaví pozici ukazatele myši.

```
procedure SetPosition( AX, AY: Integer ); virtual;
```

Parametry:

AX	X-ová souřadnice ukazatele myši.
AY	Y-ová souřadnice ukazatele myši.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Ve třídě **TCursor** je metoda **SetPosition** definovaná jako prázdná. Potomci třídy musí tuto metodu předefinovat.

7.3.12.2. Metoda TCursor.Show

Metoda **Show** zobrazí kurzor myši na displeji.

```
procedure Show; virtual;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Metody **Show** a **Hide** je možné volat rekurzivně. Pro zobrazení kurzoru po n-tém volání metody **Hide** je nutné zavolat metodu Show n-krát.

Ve třídě **TCursor** je metoda Show definovaná jako prázdná. Potomci třídy musí tuto

metodu předefinovat.

7.3.12.3. Metoda TCursor.Hide

Metoda **Hide** skryje kurzor myši.

```
procedure Hide; virtual;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Metody **Show** a **Hide** je možné volat rekurzivně. Pro zobrazení kurzoru po n-tém volání metody **Hide** je nutné zavolat metodu **Show** n-krát.

Ve třídě **TCursor** je metoda **Hide** definovaná jako prázdná. Potomci třídy musí tuto metodu předefinovat.

7.3.13. Třída T8BPPCursor

Třída T8BPPCursor plně implementuje ukazatel myši pro kreslící povrchy s osmi bity na pixel. Třída předefinovává metody předka **SetPosition**, **Show** a **Hide**.

```
P8BPPCursor = ^T8BPPCursor;
T8BPPCursor = object( TCursor )
public
  constructor Init( ADispDriver: PDisplayDriver );
  destructor  Done; virtual;

  procedure SetPosition( AX, AY: Integer ); virtual;
  procedure Show; virtual;
  procedure Hide; virtual;
end;
```

7.3.13.1. Konstruktor T8BPPCursor.Init

Konstruktor **Init** inicializuje instanci třídy.

```
constructor Init( ADispDriver: PDisplayDriver );
```

Parametry:

ADispDriver Odkaz na ovladač displeje.

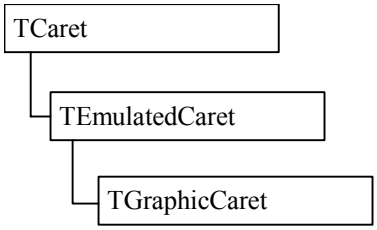
Návratové hodnoty:

Konstruktor nevrací žádnou hodnotu.

Poznámky:

7.3.14. Třída TCaret

Abstraktní třída **TCaret** definuje obecné rozhraní textového kurzoru. Textový kurzor je malý blikající obdélník v místě vkládání nebo přepisování znaku zpravidla v editační řádce.




```
PCaret = ^TCaret;
TCaret = object( TObject )
public
  procedure Show( APosition, ASize: TPoint; const AClip: TRect );
              virtual;
  procedure Hide; virtual;
  procedure SetBlinkPeriod( APeriod: Integer ); virtual;
  procedure InternalShow; virtual;
  procedure InternalHide; virtual;
  procedure Tick; virtual;
end;
```

7.3.14.1. Metoda TCaret.Show

Metoda **Show** zobrazí textový kurzor na zadané pozici.

```
procedure Show( APosition, ASize: TPoint; const AClip: TRect );
            virtual;
```

Parametry:

APosition	Souřadnice kurzoru (levý horní roh obdélníku kurzoru)
ASize	Rozměry kurzoru v pixelech.
AClip	Ořezávací oblast kurzoru. Mimo tuto oblast nebude kurzor zobrazen.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Veškeré souřadnice zadávané v parametrech jsou souřadnice displeje. Metody **Show** a **Hide** nelze volat vnořeně. Přímo ve třídě **TCaret** je metoda **Show** prázdná. Potomci třídy jí musí předefinovat.

Při zavolání metody **Show** je restartován časovač blikání kurzoru, pokud byl změněn alespoň jeden parametr kurzoru (pozice, rozměry), tak aby nový stav kurzoru byl

okamžitě vidět.

7.3.14.2. Metoda TCaret.Hide

Metoda **Hide** skryje textový kurzor.

```
procedure Hide; virtual;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Metody **Show** a **Hide** nelze volat vnořeně. Přímo ve třídě **TCaret** je metoda **Hide** prázdná. Potomci třídy jí musí předefinovat.

7.3.14.3. Metoda TCaret.SetBlinkPeriod

Metoda **SetBlinkPeriod** slouží k nastavení periody blikání kurzoru.

```
procedure SetBlinkPeriod( APeriod: Integer ); virtual;
```

Parametry:

APeriod	Perioda blikání kurzoru v milisekundách. V případě, že je parametr nastaven na hodnotu nula, je blikání vypnuto.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Přímo ve třídě **TCaret** je metoda **SetBlinkPeriod** prázdná. Potomci třídy jí musí předefinovat.

7.3.14.4. Metoda TCaret.InternalShow

Metoda **InternalShow** zabezpečuje zobrazení kurzoru po ukončení vykreslování na displej. Jedná se o speciální interní metodu.

```
procedure InternalShow; virtual;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Přímo ve třídě **TCaret** je metoda **InternalShow** prázdná. Potomci třídy jí musí předefinovat.

7.3.14.5. Metoda TCaret.InternalHide

Metoda **InternalHide** zabezpečuje skrytí kurzoru před jakýmkoli vykreslováním na displej. Jedná se o speciální interní metodu.

```
procedure InternalHide; virtual;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Přímo ve třídě **TCaret** je metoda **InternalHide** prázdná. Potomci třídy jí musí předefinovat.

Dvojice metody **InternalHide** a **InternalShow** slouží ke skrytí a zobrazení kurzoru před začátkem a po ukončení vykreslování na kreslící povrch. Jsou volány z metod **BeginDraw** a **EndDraw** třídy **TDisplayDriver** (viz. kapitoly 7.3.7.11 a 7.3.7.12). Na rozdíl od metod **Hide** a **Show** nerestartují časovač blikání a po změně kreslícího povrchu nevolají metodu **SurfaceChanged** třídy **TDisplayDriver**.

7.3.14.6. Metoda TCaret.Tick

Metoda **Tick** provádí jeden krok automatu kurzoru.

```
procedure Tick; virtual;
```

Parametry:

Metoda nemá žádné parametry.

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Metoda **Tick** obvykle provádí pouze automat řešící blikání kurzoru. Přímo ve třídě **TCaret** je metoda **Tick** prázdná. Potomci ji mohou v případě potřeby předefinovat.

7.3.15. Třída TEmulatedCaret

Abstraktní třída **TEmulatedCaret** je bázovou třídou pro všechny emulované textové kurzory, tj. kurzory pro displeje, které nepodporují vlastní hardwarový kurzor.Třída zajišťuje automatické překreslování a blikání kurzoru, tak jak je potřeba. Potomci třídy musí předefinovat pouze jednu metodu (**DrawCaret**), tak aby vykreslila kurzor požadovaného tvaru na kreslící povrch.

```
PEmulatedCaret = ^TEmulatedCaret;
TEmulatedCaret = object( TCaret )
public
  DispDriver  : PDisplayDriver;
  Clip        : TRect;
  Position    : TPoint;
  Size        : TPoint;

  constructor Init( ADispDriver : PDisplayDriver );

  procedure Show( APosition, ASize: TPoint; const AClip: TRect );
            virtual;
  procedure Hide; virtual;
  procedure SetBlinkPeriod( APeriod: Integer ); virtbual;
  procedure InternalShow; virtual;
  procedure InternalHide; virtual;
  procedure Tick; virtual;

  procedure DrawCaret( AShow: Boolean ); virtual;
end;
```

Třída **TEmulatedCaret** předefinovává metody **Show**, **Hide**, **SetBlinkPeriod**, **InternalShow**, **IntenalHide** a **Tick** předka. Popis těchto funkcí je uveden v kapitole 7.3.14.

7.3.15.1. Položka TEmulatedCaret.DispDriver

Položka **DispDriver** obsahuje odkaz na ovladač displeje, na kterém se kurzor vykresluje. Položka je nastavena v konstruktoru **Init** a je určena pouze pro čtení.

```
DispDriver : PDisplayDriver;
```

Položku **DispDriver** využívá metoda **DrawCaret** (viz. kapitola 7.3.15.5).

7.3.15.2. Položka TEmulatedCaret.Clip

Položka **Clip** obsahuje ořezávací obdélník pro vykreslení textového kurzoru. Položka je nastavována metodou **Show** (viz. kapitola 7.3.14.1).

```
Clip : TRect;
```

Položku **Clip** využívá metoda **DrawCaret** (viz. kapitola 7.3.15.5).

7.3.15.3. Položka TEmulatedCaret.Position

Položka **Position** obsahuje aktuální pozici textového kurzoru (levý horní roh obdélníku). Položka je nastavována metodou **Show** (viz. kapitola 7.3.14.1).

```
Position : TPoint;
```

Položku **Position** využívá metoda **DrawCaret** (viz. kapitola 7.3.15.5).

7.3.15.4. Položka TEmulatedCaret.Size

Položka **Size** obsahuje aktuální rozměry textového kurzoru. Položka je nastavována metodou **Show** (viz. kapitola 7.3.14.1).

```
Size : TPoint;
```

Položku **Position** využívá metoda **DrawCaret** (viz. kapitola 7.3.15.5).

7.3.15.5. Metoda TEmulatedCaret.DrawCaret

Metoda **DrawCaret** zajišťuje vykreslení nebo smazání kurzoru.

```
procedure DrawCaret( AShow: Boolean ); virtual;
```

Parametry:

AShow	Parametr určuje zda se má kurzor vykreslit (hodnota True) nebo smazat (hodnota False).

Návratové hodnoty:

Metoda nevrací žádnou hodnotu.

Poznámky:

Úkolem metody **DrawCaret** je vykreslení obdélníku reprezentující kurzor na pozici danou položkou **Position** (viz. kapitola 7.3.15.3) o rozměrech daných položkou **Size** (viz. kapitola 7.3.15.4). Vykreslený obdélník by měl být ořezán podle nastavení položky **Clip** (viz. kapitola 7.3.15.2). Pokud je nastaven parametr AShow na hodnotu True, musí metoda **DrawCaret** vykreslit kurzor. Pokud je parametr AShow nastaven na hodnotu False, musí metoda **DrawCaret** kurzor smazat a obnovit pozadí pod kurzorem.

Přímo ve třídě **TEmulatedCaret** je metoda **DrawCaret** prázdná. Potomci třídy musí tuto metodu předefinovat.

7.3.16. Třída TGraphicCaret

Třída **TGraphicCaret** implementuje standardní emulovaný kurzor pro všechny grafické displeje.

```
PGraphicCaret = ^TGraphicCaret;
TGraphicCaret = object( TEmulatedCaret )
public
  constructor Init( ADispDriver: PDisplayDriver );
  procedure DrawCaret( AShow: Boolean ); virtual;
end;
```

Třída předefinovává konstruktor, který nastaví počáteční stav kurzoru na souřadnice [0,0] o velikosti [2,8]. Kurzor je na počátku skrytý. Předefinovaná metoda **DrawCaret** vykresluje kurzor jako obdélník pomocí metody **FillRect** třídy **TDrawSurface** (viz. kapitola 7.3.9.12), s parametry Color a ROP nastavenými na hodnoty clWhite a ropXor.

7.4. Funkce

7.4.1. Procedura AssignKeyCode

Procedura **AssignKeyCode** nastaví strukturu **TEvent** (viz. kapitola 7.2.1) tak, aby obsahovala událost stisku zadané klávesy.

```
procedure AssignKeyCode( var AEvent: TEvent; AKeyCode: Word );
```

Parametry:

AEvent	Parametr AEvent bude po provedení funkce vyplněn událostí stisku klávesy.
AKeyCode	Kód klávesy (viz. kapitola 7.1.2).

Poznámky:

Procedura vyplní položky Code, KeyCode, VirtKey a CharCode struktury **TEvent**.

7.4.2. Funkce GetTickCount

Funkce **GetTickCount** vrací počet tiků přerušení IRQ0 od resetu počítače modulo počet tiků přerušení za jeden den.

```
function GetTickCount: Longint;
```

Parametry:

Funkce nemá žádné parametry.

Návratové hodnoty:

Funkce **GetTickCount** vrací počet tiků přerušení IRQ0 od resetu počítače modulo počet tiků přerušení za jeden den.

Poznámky:

7.4.3. Funkce GetCheckSum

Funkce GetCheckSum počítá kontrolní součet bloku dat.

```
function GetCheckSum( Data: Pointer; Size: Word ): Word;
```

Parametry:

Data	Ukazatel na začátek bloku dat.
Size	Délka dat v bajtech.

Návratové hodnoty:

Metoda vrací kontrolní součet bloku dat.

Poznámky:

Pokud součástí bloku dat je i položka samotného kontrolního součtu, měla by být před voláním funkce **GetCheckSum** vynulována a nastavena až po návratu funkce na vrácenou hodnotu. Při kontrole takto nastavené struktury lze využít toho, že funkce **GetCheckSum** vrátí vždy hodnotu nula.